µC/OS-II

Micro-Controller Operating System II

Core: v2.52

Port: Atmel AVR-GCC v270603
(Julius Luukko)

Gábor Naszály
BME – MIT
naszaly@mit.bme.hu

ver.: 2012a

1. The story of µC/OS

- **Developer:** Jean J. Labrosse
- **Motivation:** they needed a RT kernel for an application
  - Kernel „A”: already known, stable, but too expensive
  - Kernel „B”: unknown but cheaper
    - They had decided to buy this one
    - They spent 2 month just for creating some tasks
    - As it came out, they were one of the first customers of this kernel, and it hadn’t been tested well

1. The story of µC/OS

- After all they bought kernel “A” too
  - They made it work within two days
  - But they had found a bug after 3 months
  - It was a pity because the warranty became void after 90 days
  - They tried to convince the maker of the kernel to fix the bug for free (because by finding a bug they made a favor to them)
  - Unfortunately they failed to convince them, so they paid for the bug fix too!
- Certainly they became extremely furious and finished their product with a huge overdue

1. The story of µC/OS

- Jean J. Labrosse

> ”Well, it can’t be that difficult to write a kernel. All it needs to do is save and restore processor registers.”

- Working at nights and on weekends they made a new kernel
- After a year they reached the level of kernel “A”
- They didn’t want to found a new company (there was already about 50 kernels on the market)

1. The story of µC/OS

- He wanted to publish an article in C User’s Journal instead. But he was refused for the following reasons:
  - The article was too long
  - ”Another kernel article?”
- Then he called the Embedded Systems Programming magazine:
  - At the beginning he was refused for the same reasons
  - But calling the editor three times a week he eventually managed to publish the article
  - In that year (1992) it was the most read article of the magazine

1. The story of µC/OS

- Dr. Bernard Williams, publisher of C User's Journal, called him shortly after about the article:

> **Jean J. Labrosse:** "Don't you think you are a little bit late with this? The article is being published in ESP."
>
> **Dr. Bernard Williams:** "No, No, you don't understand, because the article is so long, I want to make a book out of it."

- → Book: *µC/OS, The Real-Time Kernel*
- → Conferences → Success → Company

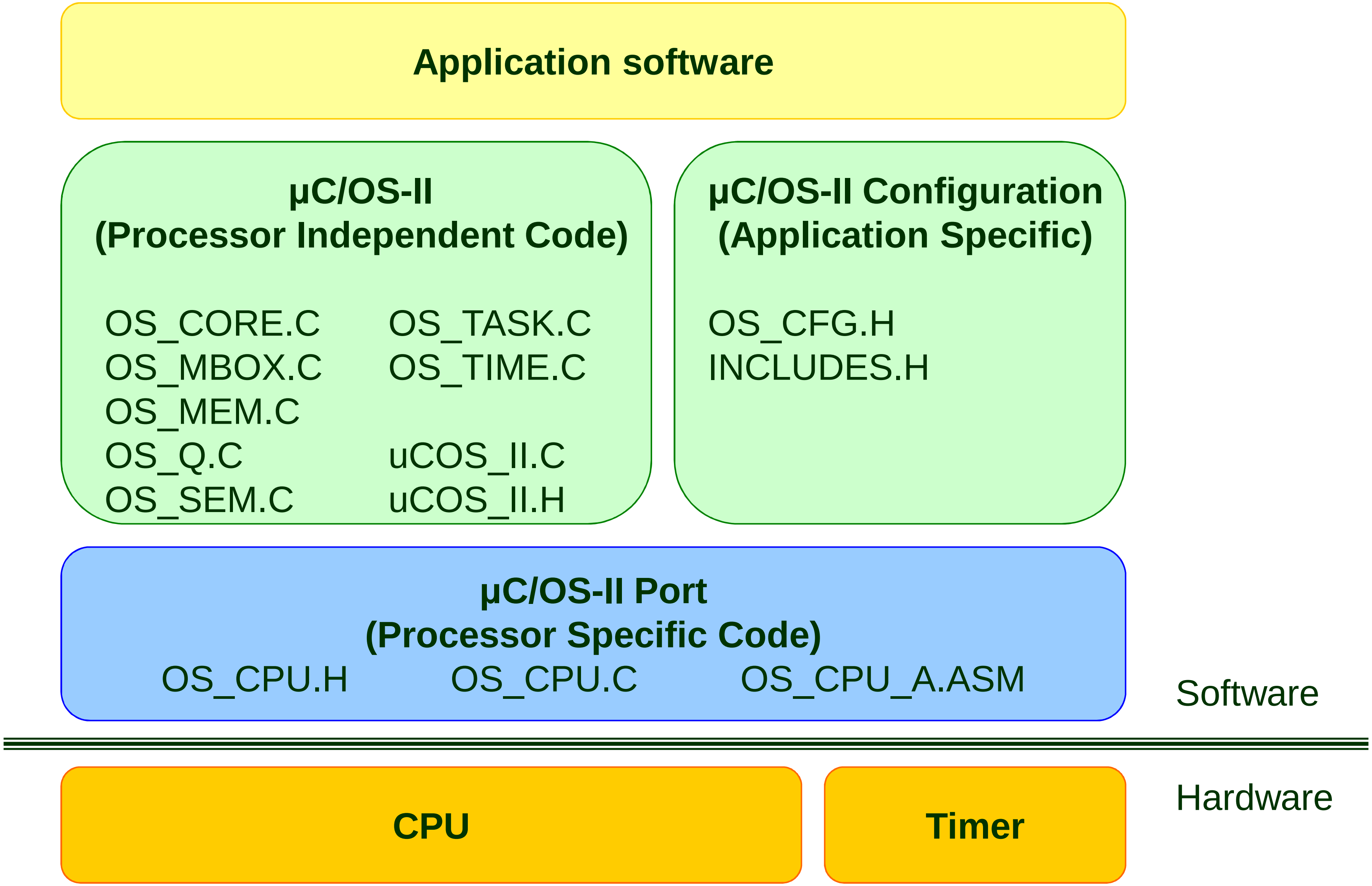
2. Features of μC/OS

- The source code is accessible
- Portable (processor dependent parts are separated)
- Scalable
- Multi-tasking
- Preemptive
- Deterministic
- Every task can have a different size task
- OS services: mailbox, queue, semaphore, fixed/sized memory partitions, time related functions, etc.
- Interrupt management (255 level nesting)
- Robust and reliable

2. Features of µC/OS

- Very well documented (µC/OS-II, The Real-Time Kernel about 300 pages long)
- For educational purposes the kernel is free
- Additional packages:
  - TCP-IP (Protocol Stack)
  - FS (Embedded File System)
  - GUI (Embedded Graphical User Interface)
  - USB Device (Universal Serial Bus Device Stack)
  - USB Host (Universal Serial Bus Host Stack)
  - FL (Flash Loader)
  - Modbus (Embedded Modbus Stack)
  - CAN (CAN Protocol Stack)
  - BuildingBlocks (Embedded Software Components)
  - Probe (Real-Time Monitoring)

3. The structure of µC/OS



Application software

µC/OS-II
(Processor Independent Code)

OS_CORE.C
OS_MBOX.C
OS_MEM.C
OS_Q.C
OS_SEM.C

OS_TASK.C
OS_TIME.C

uCOS_II.C
uCOS_II.H

µC/OS-II Configuration
(Application Specific)

OS_CFG.H
INCLUDES.H

µC/OS-II Port
(Processor Specific Code)

OS_CPU.H OS_CPU.C OS_CPU_A.ASM

Software

Hardware

CPU

Timer

4. Configuring µC/OS (OS_CFG.H)

The OS can be scaled using **#define** statements
in the configuration header file.

```
                         /* ---------------------- Miscellaneous ----------------------- */

#Define OS_ARG_CHK_EN           1    /* enable (1) or disable (0) argument checking              */
#Define OS_CPU_HOOKS_EN         0    /* uc/os-ii hooks are found in the processor port files     */
#Define OS_LOWEST_PRIO          63   /* defines the lowest priority that can be assigned         */
#Define OS_MAX_EVENTS           20   /* max. Number of event control blocks in your application  */
#Define OS_MAX_FLAGS            5    /* max. Number of event flag groups in your Application     */
#Define OS_MAX_MEM_PART         10   /* max. Number of memory partitions                         */
#Define OS_MAX_QS               5    /* max. Number of queue control blocks in your Application  */
#Define OS_MAX_TASKS            32   /* max. Number of tasks in your application                 */
#Define OS_SCHED_LOCK_EN        1    /*  include code for osschedlock() and Osschedunlock()      */
#Define OS_TASK_IDLE_STK_SIZE   512  /* idle task stack size                                     */
#Define OS_TASK_STAT_EN         1    /* enable (1) or disable(0) the statistics task             */
#Define OS_TASK_STAT_STK_SIZE   512  /* statistics task stack size                               */
#Define OS_TICKS_PER_SEC        200  /* set the number of ticks in one second                    */
```

4. Configuring µC/OS (OS_CFG.H)

```
                           /* ---------------------- EVENT FLAGS ----------------------- *
#define OS_FLAG_EN              0     /* Enable (1) or Disable (0) code generation for EVENT FLAGS */
#define OS_FLAG_WAIT_CLR_EN     1     /* Include code for Wait on Clear EVENT LAGS   */
#define OS_FLAG_ACCEPT_EN       1     /*    Include code for OSFlagAccept()          */
#define OS_FLAG_DEL_EN          1     /*    Include code for OSFlagDel()             */
#define OS_FLAG_QUERY_EN        1     /*    Include code for OSFlagQuery()           */


                           /* ----------------------- SEMAPHORES ----------------------- */
#define OS_SEM_EN               0     /* Enable (1) or Disable (0) code generation for SEMAPHORES */
#define OS_SEM_ACCEPT_EN        1     /*   Include code for OSSemAccept()            */
#define OS_SEM_DEL_EN           1     /*   Include code for OSSemDel()               */
#define OS_SEM_QUERY_EN         1     /*   Include code for OSSemQuery()             */


                       /* ---------------- MUTUAL EXCLUSION SEMAPHORES --------------- */
#define OS_MUTEX_EN             0     /* Enable (1) or Disable (0) code generation for MUTEX   */
#define OS_MUTEX_ACCEPT_EN      1     /*    Include code for OSMutexAccept()         */
#define OS_MUTEX_DEL_EN         1     /*    Include code for OSMutexDel()            */
#define OS_MUTEX_QUERY_EN       1     /*    Include code for OSMutexQuery()          */
```

4. Configuring µC/OS (OS_CFG.H)

```
                          /* -------------------- MESSAGE MAILBOXES --------------------- */
#define OS_MBOX_EN               1    /* Enable (1) or Disable (0) code generation for MAILBOXES */
#define OS_MBOX_ACCEPT_EN        1    /* Include code for OSMboxAccept()          */
#define OS_MBOX_DEL_EN           0    /* Include code for OSMboxDel()             */
#define OS_MBOX_POST_EN          1    /* Include code for OSMboxPost()            */
#define OS_MBOX_POST_OPT_EN      0    /* Include code for OSMboxPostOpt()         */
#define OS_MBOX_QUERY_EN         0    /* Include code for OSMboxQuery()           */

                          /* ---------------------- MESSAGE QUEUES ---------------------- */
#define OS_Q_EN                  1    /* Enable (1) or Disable (0) code generation for QUEUES */
#define OS_Q_ACCEPT_EN           1    /* Include code for OSQAccept()             */
#define OS_Q_DEL_EN              1    /* Include code for OSQDel()                */
#define OS_Q_FLUSH_EN            1    /* Include code for OSQFlush()              */
#define OS_Q_POST_EN             1    /* Include code for OSQPost()               */
#define OS_Q_POST_FRONT_EN       1    /* Include code for OSQPostFront()          */
#define OS_Q_POST_OPT_EN         1    /* Include code for OSQPostOpt()            */
#define OS_Q_QUERY_EN            1    /* Include code for OSQQuery()              */

                          /* -------------------- MEMORY MANAGEMENT -------------------- */
#define OS_MEM_EN                0    /* Enable (1) or Disable (0) code gen.for MEM.MANAGER */
#define OS_MEM_QUERY_EN          1    /* Include code for OSMemQuery()            */
```

4. Configuring µC/OS (OS_CFG.H)

```
                        /* --------------------- TASK MANAGEMENT ---------------------- */
#define OS_TASK_CHANGE_PRIO_EN  0  /* Include code for OSTaskChangePrio()                      */
#define OS_TASK_CREATE_EN       0  /* Include code for OSTaskCreate()                          */
#define OS_TASK_CREATE_EXT_EN   1  /* Include code for OSTaskCreateExt()                       */
#define OS_TASK_DEL_EN          0  /* Include code for OSTaskDel()                             */
#define OS_TASK_SUSPEND_EN      0  /* Include code for OSTaskSuspend() and OSTaskResume()      */
#define OS_TASK_QUERY_EN        0  /* Include code for OSTaskQuery()                           */

                        /* --------------------- TIME MANAGEMENT ---------------------- */
#define OS_TIME_DLY_HMSM_EN     1  /* Include code for OSTimeDlyHMSM()                         */
#define OS_TIME_DLY_RESUME_EN   0  /* Include code for OSTimeDlyResume()                       */
#define OS_TIME_GET_SET_EN      0  /* Include code for OSTimeGet() and OSTimeSet()             */

typedef INT16U OS_FLAGS;           /* Date type for event flag bits (8, 16 or 32 bits)         */
```

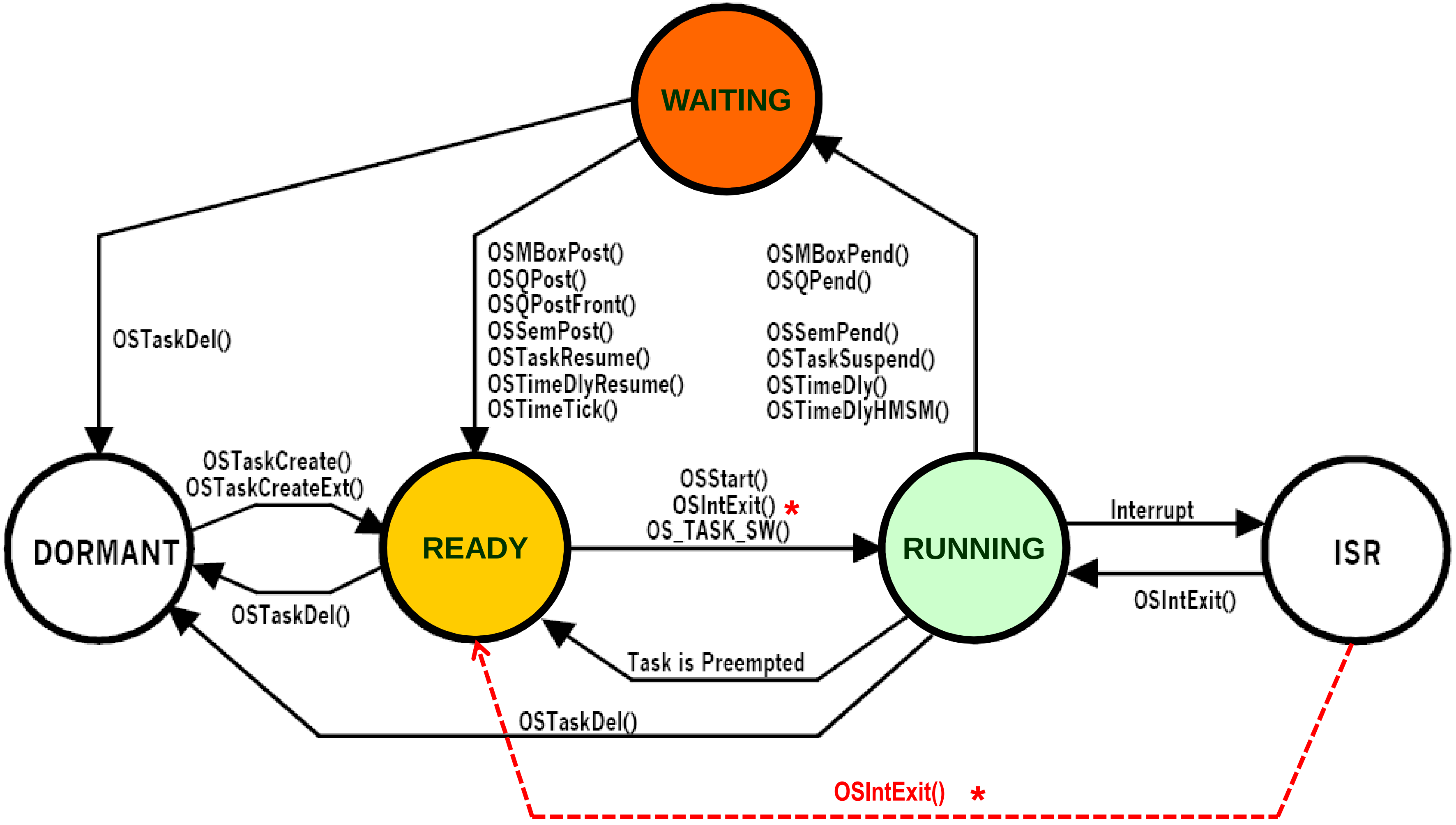
4. Configuring µC/OS (OS_CFG.H)

Conditional preprocessor directives refer to the **#define** statements.

```
#if OS_SEM_ACCEPT_EN > 0
INT16U  OSSemAccept (OS_EVENT *pevent) {
   INT16U     cnt;
#if OS_CRITICAL_METHOD == 3        /* Allocate storage for CPU status register  */
   OS_CPU_SR  cpu_sr = 0;
#endif

#if OS_ARG_CHK_EN > 0
   if (pevent == (OS_EVENT *)0) {      /* Validate 'pevent'                         */
      return (0);
   }
#endif
   …
}
#endif
```

5. µC/OS task states



5. µC/OS task states

- Two special states:
  - **DORMANT:** if the task is in the memory, but the scheduler doesn’t administer it (eighter because it hasn’t even been created (by calling **`OSTaskCreate()`**) or has already been deleted (by calling **`OSTaskDel()`**)
  - **ISR:** an interrupt has occured, thus the processor suspends the execution of the task code and jumps to an interrupt service routine (ISR). In one of the cases the ISR returns to the task (where it has been suspended). However there are cases when an ISR makes a higher priority task ready to run. In this case - after return from the ISR - the scheduler switches to the higher priority task. The original task becomes READY.

5. µC/OS task states

- If there is no task ready to run, then the OS executes the idle task (**`OSTaskIdle()`**).

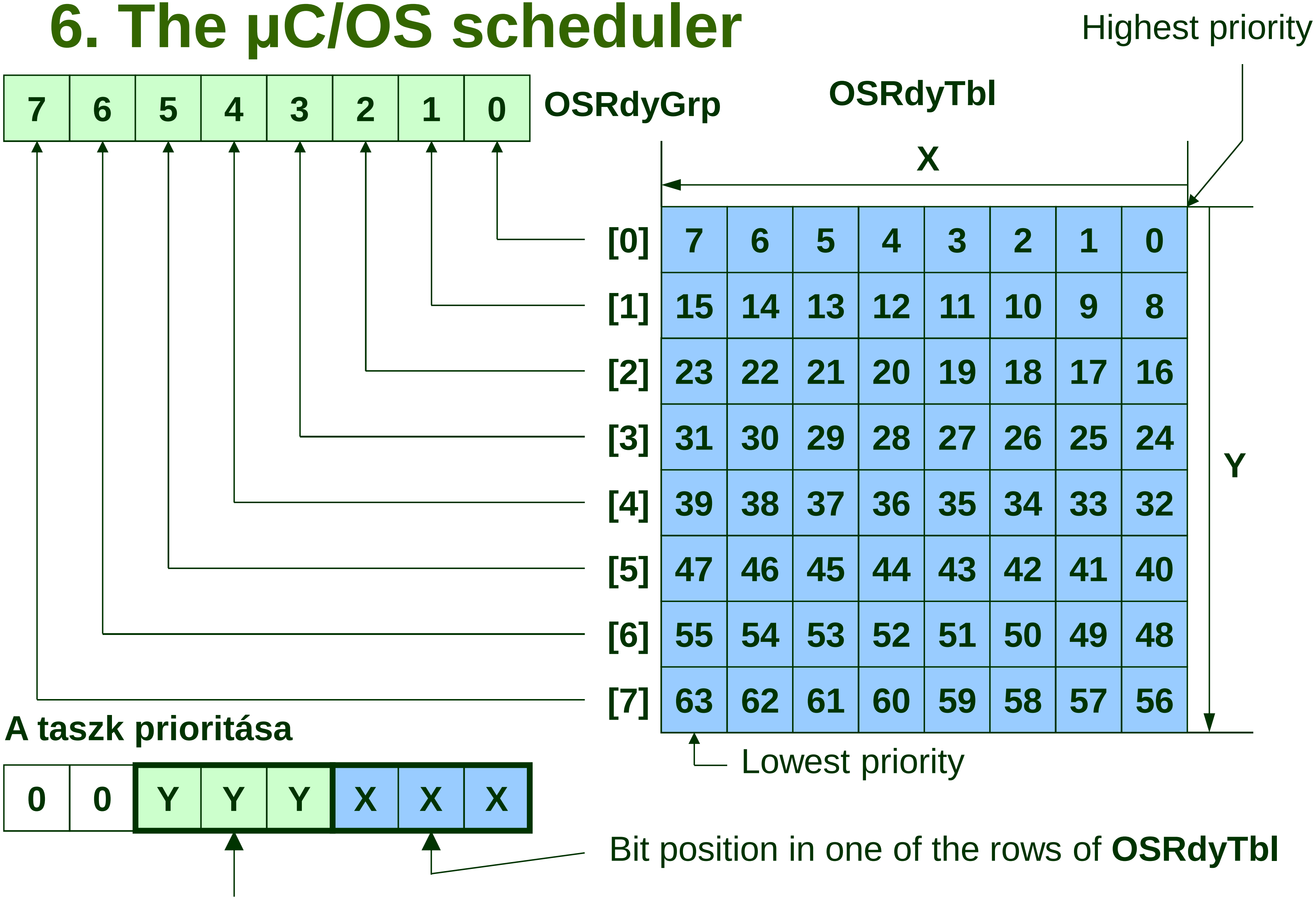
6. The µC/OS scheduler

- The tasks are represented by bits in a **2D bitmap** structure. A logical one means that the given task is ready to run. Each bit position refers to a unique priority.

6. The µC/OS scheduler

- \+ Injecting bits is fast (and independent of the number of the tasks ready to run).
- \+ It is also fast to find the highest priority task ready to run.
- – The tasks must have unique priorities (so we can’t use round-robin / time-slicing scheduling)
- – We need some lookup tables (consuming relatively much data memory)

6. The µC/OS scheduler



Highest priority

OSRdyGrp

7	6	5	4	3	2	1	0

OSRdyTbl

[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

Lowest priority

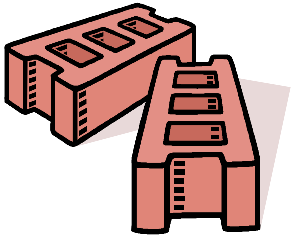
A taszk prioritása

0	0	Y	Y	Y	X	X	X

Bit position in one of the rows of **OSRdyTbl**

Designates one of the rows of **OSRdyTbl** and the corresponding bit in **OSRdyGrp**

6. The µC/OS scheduler



- Building blocks of scheduling:
  1. Marking a task as not ready to run
  2. Marking a task as ready to run
  3. Finding the highest priority task among the ones that are ready to run
  4. Switching context

6. The µC/OS scheduler

- Where are these blocks used?
  1. **Marking a task as NOT ready to run:** in system calls causing a task to wait (eg. **`OSSemPend()`**, **`OSMBoxPend()`**, **`OSTimeDly()`**). These calls do some unique operation (e.g. trying to lock a semaphore), then (if it is needed) mark the task as NOT ready to run and call the scheduler function.

6. The µC/OS scheduler

2. **Marking a task as READY to run:** in system calls causing some event which others are maybe awaiting for (eg. **`OSSemPost()`**, **`OSMBoxPost()`**) or the elapse of a given time. They do some unique operation (e.g. releasing a semaphore), then (if it is needed) mark a task as READY to run and call the scheduler function.

6. The µC/OS scheduler

3. **Finding the highest priority task among the ones that are ready to run:** this is located in the scheduler function.
4. **Switching context:** if this priority represents a different task than the one currently running, the scheduler function also does the context switching.

6. The µC/OS scheduler

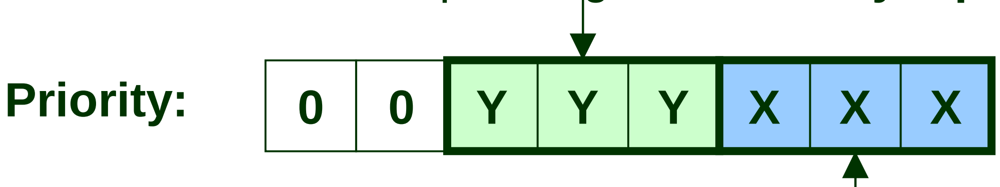
2. Marking a task as READY to run

The bits designated by the task's priority has to be set in `OSRdyGrp` and in `OSRdyTbl`.

```
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



Designates one of the rows of **OSRdyTbl** and the corresponding bit in **OSRdyGrp**

Bit position in one of the rows of **OSRdyTbl**

prio: the task's priority
prio >> 3: YYY
prio & 0x07: XXX

6. The µC/OS scheduler

1. Marking a task as NOT READY to run

We have to clear the bit designated by the task's priority from the corresponding row of **`OSRdyTbl`**. If this results in no more ready to run task in the given row, we have to clear a bit from **`OSRdyGrp`** corresponding to the given row.

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

6. The µC/OS scheduler

3. Finding the highest priority task among the ones ready to run

Which equals to finding the 1 in the top rightmost position. (Which can be divided into two ‘find the rightmost bit in a byte’ operation. First in OSRdyGrp, then in one of OSRdyTbl rows.)

For example:

Hexa	**Binary**								**Rightmost bit set**
	7.	**6.**	**5.**	**4.**	**3.**	**2.**	**1.**	**0.**	
0x2C	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**2.**

6. The µC/OS scheduler

```
INT8U  const  OSUnMapTbl[256] = {
    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 */
};
```

6. The µC/OS scheduler

At first we search for the rightmost 1 in OSRdyGrp designating the topmost row in OSRdyTbl (YYY value), then we search for the rightmost 1 in that row (XXX value). After that the highest priority can be calculated easily: YYY * 8 + XXX.

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

6. The µC/OS scheduler

4. Switching context

1. Saving current context:
   - *Saving registers*
   - Saving stack pointer

2. Restoring new context:
   - Restoring stack pointer
   - *Restoring registers*

Switching context is processor dependent!

6. The µC/OS scheduler

- The scheduler is implemented as a function (which is called by other OS functions). For example, if we want to lock a semaphore (which is not free at the moment) then calling **`OSSemPend()`** also administers that our task is no more ready to run, then calls the scheduler. The scheduler selects the task with the highest priority among the remainder ones ready to run, then do the context switch to it.

6. The µC/OS scheduler

```
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();
}
```

The scheduling code is a critical section: we need to disable interrupts before and reenable them after (CPU dependent). Then we ensure that the scheduler runs only when it hasn’t been locked and when not called from ISR. Then the scheduler search for the highest priority task ready to run. Switching context occurs only when this priority doesn’t equals to the priority of the currently running task.

6. The µC/OS scheduler

```
if (OSPrioHighRdy != OSPrioCur) {
    OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
    OSCtxSwCtr++;
    OS_TASK_SW();
}
```

At first we retrieve the TCB (Task Control Block) belonging to the task which we want to switch to. (The TCB holds the basic information belonging to a given task: its priority, its stack pointer (previously saved), etc.) Then we increment a variable counting context switches purely for statistical purposes. The actual instructions needed for switching context (saving then restoring registers, stack pointer) is done in processor dependent assembly code.

6. The µC/OS scheduler

Main fileds in the TCB structure:

```
typedef struct os_tcb {
  OS_STK  *OSTCBStkPtr; // Pointer to top of stack
  ...
  INT16U  OSTCBDly; // Ticks to delay task or timeout wait
  INT8U   OSTCBStat; // Task status
  INT8U   OSTCBPrio; // Task priority
   ...
} OS_TCB;
```

6. The µC/OS scheduler

OS_TASK_SW()

1. Saving current context:
    - *Saving registers*
    - Saving stack pointer

2. Restoring new context:
    - Restoring stack pointer
    - *Restoring registers*

The context switch is **platform dependent!**

The given example is for the **AVR ATmega128**.

```
PUSHRS
PUSHSREG

LDS     R30,OSTCBCur
LDS     R31,OSTCBCur+1
in      r28,_SFR_IO_ADDR(SPL)
ST      Z+,R28
in      r29,_SFR_IO_ADDR(SPH)
ST      Z+,R29

CALL    OSTaskSwHook
LDS     R16,OSPrioHighRdy
STS     OSPrioCur,R16

LDS     R30,OSTCBHighRdy
LDS     R31,OSTCBHighRdy+1
STS     OSTCBCur,R30
STS     OSTCBCur+1,R31
LD      R28,Z+
out     _SFR_IO_ADDR(SPL),R28
LD      R29,Z+
out     _SFR_IO_ADDR(SPH),R29

POPSREG
POPRS
```

7. OS services

- Task management
  - Create / Delete a task
  - Suspend / Resume a task
  - Change a task's priority
- Time management
  - Delay the execution of a task
  - Get / Set system time
- Memory management
  - Create a memory partition
  - Request / Release a block in a partition

7. OS services

- Semaphore management
  - Initialize a semaphore
  - Pend on a semaphore (optional timeout)
  - Accept a semaphore (non blocking)
  - Release a semaphore
- Message mailbox management
  - Same operations as for the semaphores

7. OS services

- Message queue management
  - Same operations as for the mailboxes
  - Post to the front of the queue
  - Flush the queue
- Mutex management
  - Same operations as for the semaphores
- Event flag management
  - Same operations as for the semaphores
  - Waiting for flags can be: AND, OR

8. Typical layout of a µC/OS application

```
void YourTask (void *pdata){
 for (;;) {
  /* USER CODE */
  !! Call one of uC/OS-II’s
  !! services: OSMboxPend(),
  !! OSQPend(),OSSemPend(),
  !! OSTaskDel(OS_PRIO_SELF),
  !! OSTaskSuspend(OS_PRIO_SELF),
  !! OSTimeDly(),OSTimeDlyHMSM()
  /* USER CODE */
 }
}
```

Infinite loop task

or

’’Single shot” task

```
void YourTask (void *pdata)
{
 /* USER CODE */
 OSTaskDel(OS_PRIO_SELF);
}
```

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