

Getting Started with Embedded OS

Embedded OS Implementation

Prof. Ya-Shu Chen

National Taiwan University
of Science and Technology
2020 Fall

uC/OS-2

- A tiny open-source real-time kernel
 - Memory footprint is about 20k for a fully functional kernel
 - Supporting preemptive priority-driven real-time scheduling
 - Supporting many platforms: x86, 68x, ARM, MIPS...

Getting started with uC/OS-2!

- See what a uC/OS-2 program looks like.
- Learn how to write a skeleton program for uC/OS-2.
 - How to initialize uC/OS-2?
 - How to create tasks?
 - How to use inter-task communication mechanism?

Example 1

- 13 tasks run concurrently
 - 2 internal tasks:
 - The idle task and the statistic task
 - 11 user tasks:
 - Randomly print numbers onto the screen
- Focus: System initialization and task creation

Example 1

```
• #include "includes.h"

• /*
• *****
• *
• *                               CONSTANTS
• *
• *****
• */

• #define TASK_STK_SIZE          512      /* Size of each task's stacks (# of WORDs)      */
• #define N_TASKS                10      /* Number of identical tasks                  */

• /*
• *****
• *
• *                               VARIABLES
• *
• *****
• */

• OS_STK      TaskStk[N_TASKS][TASK_STK_SIZE];      /* Tasks stacks                               */
• OS_STK      TaskStartStk[TASK_STK_SIZE];
• char        TaskData[N_TASKS];                  /* Parameters to pass to each task           */
• OS_EVENT    *RandomSem;
```



A semaphore (to be explained later)

Main()

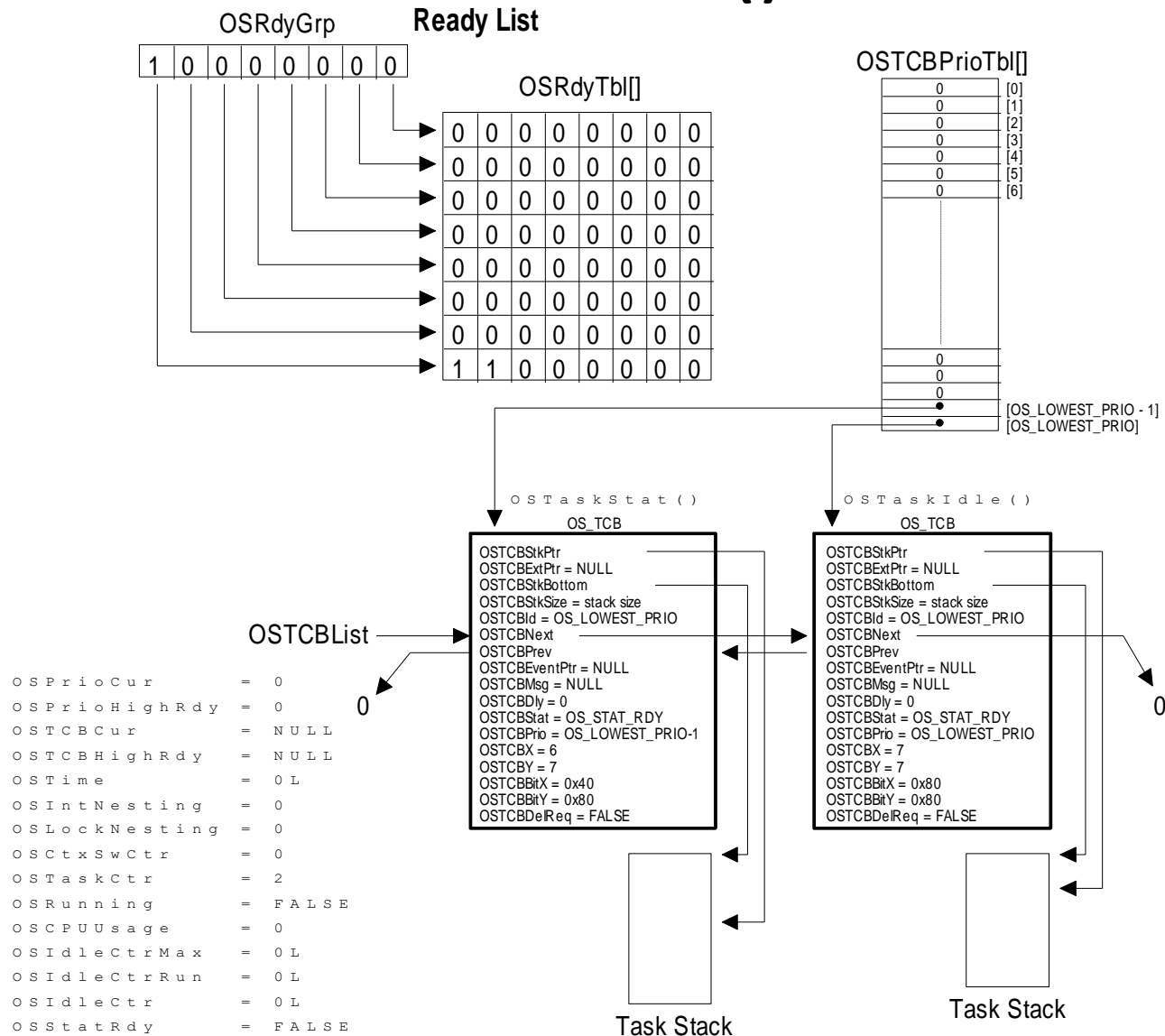
- `void main (void)`
- `{`
- `PC_DispClrScr(DISP_FGND_WHITE + DISP_BGND_BLACK);` (1)
- `OSInit();` (2)
- `PC_VectSet(uCOS, OSCtxSw);` (4)
- `RandomSem = OSSemCreate(1);` (5)
- `OSTaskCreate(TaskStart,` (6)
- `(void *)0,`
- `(void *)&TaskStartStk[TASK_STK_SIZE-1],`
- `0);`
- `OSStart();` (7)
- `}`



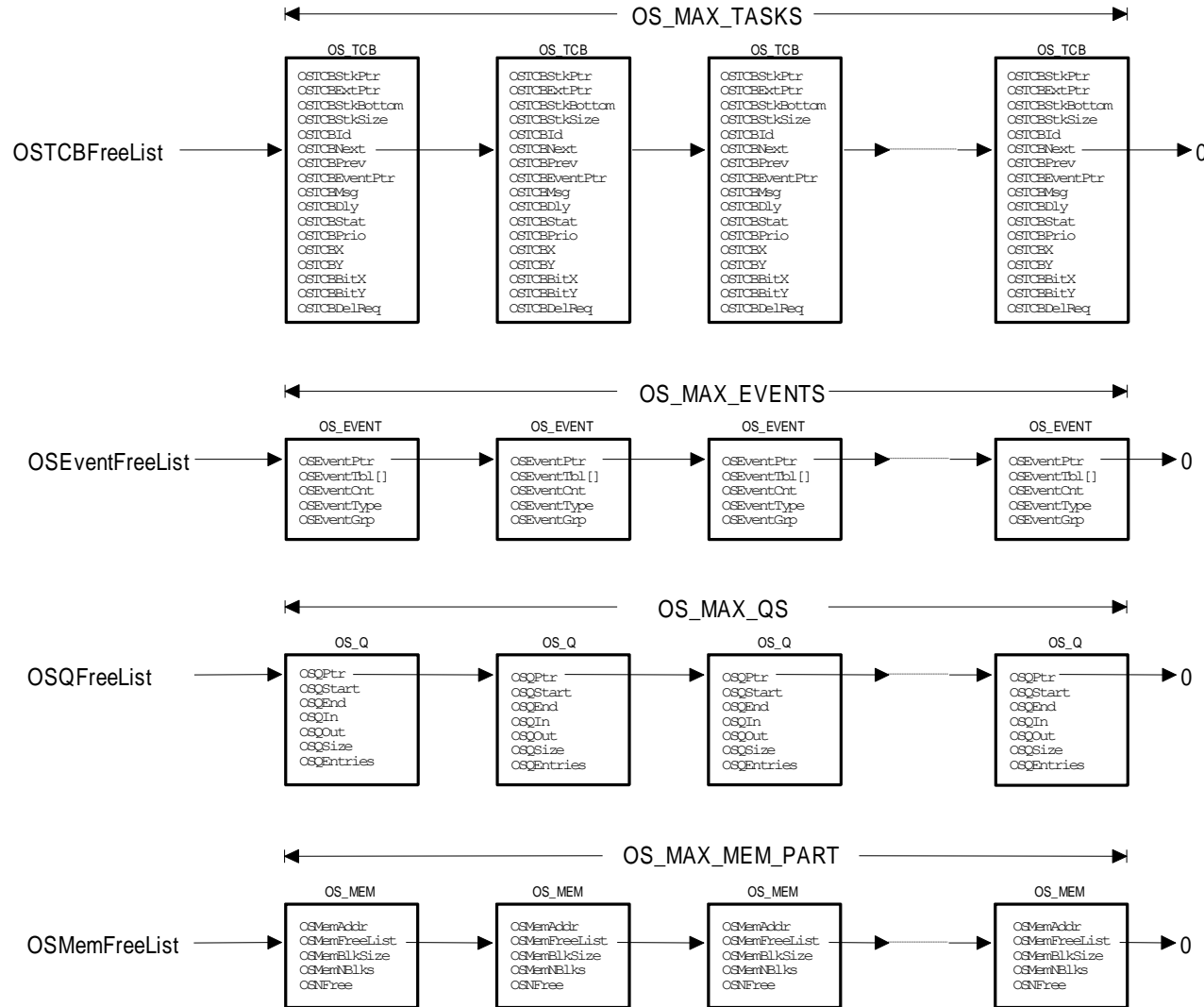
Main()

- OSinit():
 - Init internal structures of uC/OS-2.
 - Task ready list
 - Priority table
 - Task control blocks (TCB)
 - Free pool
 - Create housekeeping tasks.
 - The idle task
 - The statistics task

OSinit()



OSinit()



Main()

- PC_VectSet(uCOS,OSCtXSw)
 - Install the context switch handler
 - Interrupt # 0x80 of 80x86 family
 - Later invoked by int instruction



Main()

- OSSemCreate()
 - Create a semaphore for IPC
 - To protect non-reentrant codes and shared resources
 - The semaphore is initialized as a binary semaphore
 - For mutual exclusion
 - In this example, a semaphore is created to protect “random()” in the standard C library
 - It is non-reentrant. To be explained later.

Main()

- OSTaskCreate()
 - Create tasks with the supplied arguments
 - Tasks become “ready” after created
- Task
 - An active entity which could do some computations
 - **Priority, CPU registers, stack, text, housekeeping status**
 - uC/OS-2 allows maximum 63 tasks to be created
- uC/OS-2 picks up the highest-priority task for execution on rescheduling points
 - Clock ticks, interrupt return, and semaphore operations...
 - We shall see more in RTC ISR.



OSTaskCreate()

- **OSTaskCreate (**
 TaskStart,
 (void *)0,
 &TaskStartStk[TASK_STK_SIZE - 1],
 0
);
-
- Entry point of the task (a pointer to function)
- User-specified data
- Priority (0=highest)
- Top of Stack

Main()

- OSStart()
 - Start multitasking of uC/OS-2
 - It never returns to main()
 - uC/OS-2 is terminated if PC_DOSReturn() is called



TaskStart()

```
• void TaskStart (void *pdata)
• {
• #if OS_CRITICAL_METHOD == 3 /* Allocate storage for CPU status register */
• OS_CPU_SR cpu_sr;
• #endif
• char s[100];
• INT16S key;

• pdata = pdata; /* Prevent compiler warning */
• TaskStartDispInit(); /* Initialize the display */

• OS_ENTER_CRITICAL();
• PC_VectSet(0x08, OSTickISR); /* Install uC/OS-II's clock tick ISR */
• PC_SetTickRate(OS_TICKS_PER_SEC); /* Reprogram tick rate */
• OS_EXIT_CRITICAL();

• OSStatInit(); /* Initialize uC/OS-II's statistics */
• TaskStartCreateTasks(); /* Create all the application tasks */

• for (;;) {
• TaskStartDisp(); /* Update the display */

• if (PC_GetKey(&key) == TRUE) { /* See if key has been pressed */
• if (key == 0x1B) { /* Yes, see if it's the ESCAPE key */
• PC_DOSReturn(); /* Return to DOS */
• }
• }

• OSCtxSwCtr = 0; /* Clear context switch counter */
• OSTimeDlyHMSM(0, 0, 1, 0); /* Wait one second */
• }
• }
```



TaskStart()

- OS_ENTER(EXIT)_CRITICAL
 - Enable/disable most interrupts
 - An alternative way to accomplish mutual exclusion
 - No rescheduling is possible during the disabling of interrupts
 - Different from semaphores
 - Processor specific
 - CLI/STI (x86 real mode)
 - Interrupt descriptors (x86 protected mode)



TaskStartCreateTasks()

```
• static void TaskStartCreateTasks (void)
• {
•     INT8U i;

•     for (i = 0; i < N_TASKS; i++) {
•
•         TaskData[i] = '0' + i;

•         OSTaskCreate(
•             Task,
•             (void *)&TaskData[i],
•             &TaskStk[i][TASK_STK_SIZE - 1],
•             i + 1);
•     }
• }
```

Entry point of the
created task

Argument: character
to print

Priority

Stack



Task()

```
• void Task (void *pdata)
• {
•     INT8U x;
•     INT8U y;
•     INT8U err;

•     for (;;) {
•         OSSEmPend(RandomSem, 0, &err); /* Acquire semaphore to perform random numbers */
•         x = random(80);                /* Find X position where task number will appear */
•         y = random(16);                /* Find Y position where task number will appear */
•         OSSEmPost(RandomSem);          /* Release semaphore */
•                                         /* Display the task number on the screen */
•         PC_Dispatch(x, y + 5, *(char *)pdata, DISP_FGND_BLACK + DISP_BGND_LIGHT_GRAY);
•         OSTimeDly(1);                  /* Delay 1 clock tick */
•     }
• }
```

Semaphore operations.



Semaphores

- `OSSemPend()` / `OSSemPost()`
 - A semaphore consists of a wait list and an integer counter.
 - `OSSemPend`:
 - `Counter--;`
 - If the value of the semaphore < 0 , the task is blocked and moved to the wait list immediately.
 - A time-out value can be specified .
 - `OSSemPost`:
 - `Counter++;`
 - If the value of the semaphore ≥ 0 , a task in the wait list is removed from the wait list.
 - Reschedule if needed.

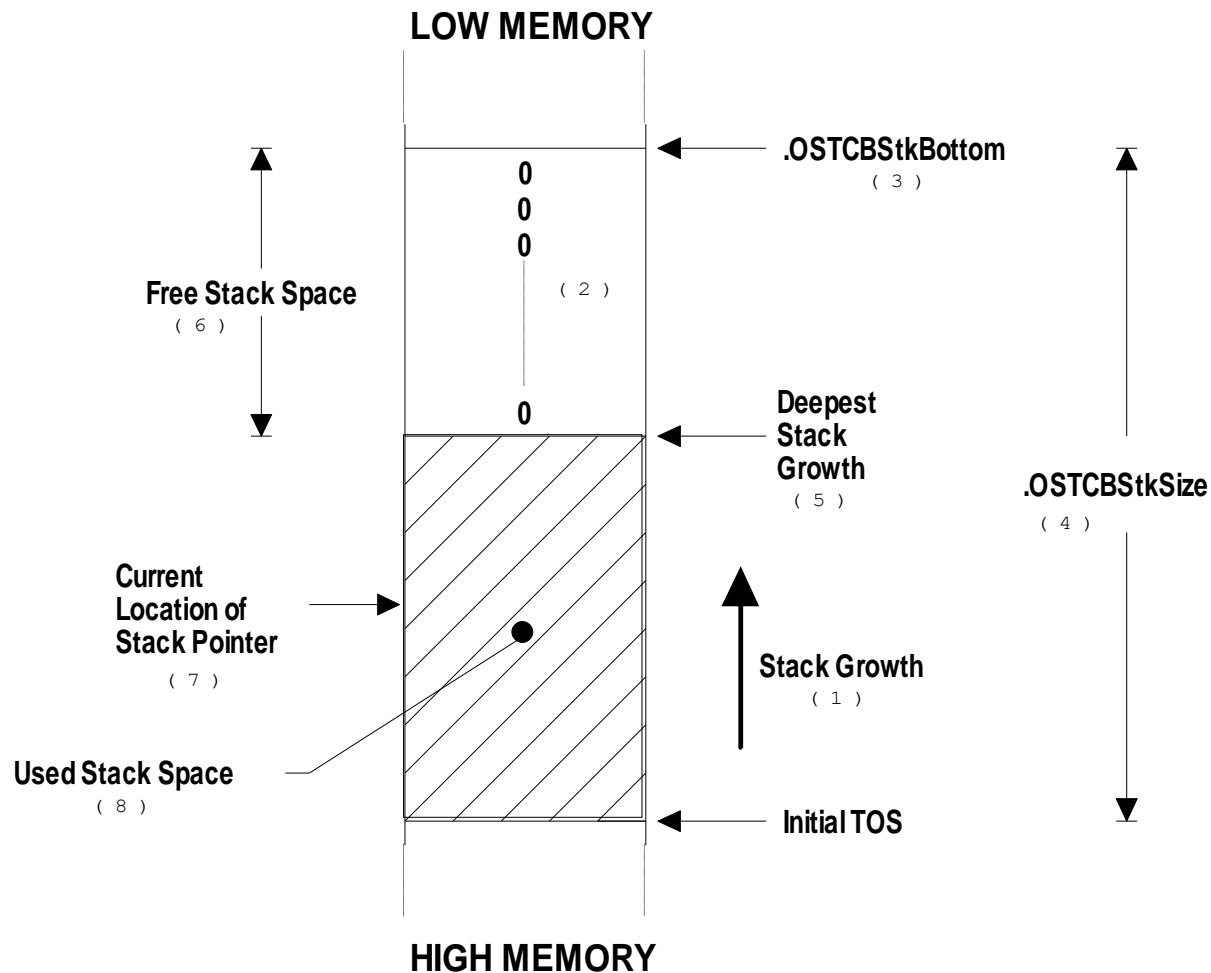
Summary: Example 1

- uC/OS-2 is initialized and started by calling OSInit() and OSStart(), respectively
- Before uC/OS-2 is started,
 - DOS status is saved by calling PC_DOSSaveReturn()
 - Context switch handler is installed by calling PC_VectSet()
 - User tasks must be created by OSTaskCreate()
- Shared resources must be protected by semaphores
 - OSSemPend(),OSSemPost()

Example 2

- Example 2 focuses on:
 - More task creation options
 - **Stack usage** of each task
 - **Floating point** operations
 - IPC via **mailboxes**

Stack Usage of a Task



```

• #define TASK_STK_SIZE 512 /* Size of each task's stacks (# of WORDs) */
• #define TASK_START_ID 0 /* Application tasks IDs */
• #define TASK_CLK_ID 1
• #define TASK_1_ID 2
• #define TASK_2_ID 3
• #define TASK_3_ID 4
• #define TASK_4_ID 5
• #define TASK_5_ID 6

• #define TASK_START_PRIO 10 /* Application tasks priorities */
• #define TASK_CLK_PRIO 11
• #define TASK_1_PRIO 12
• #define TASK_2_PRIO 13
• #define TASK_3_PRIO 14
• #define TASK_4_PRIO 15
• #define TASK_5_PRIO 16

• OS_STK TaskStartStk[TASK_STK_SIZE]; /* Startup task stack */
• OS_STK TaskClkStk[TASK_STK_SIZE]; /* Clock task stack */
• OS_STK Task1Stk[TASK_STK_SIZE]; /* Task #1 task stack */
• OS_STK Task2Stk[TASK_STK_SIZE]; /* Task #2 task stack */
• OS_STK Task3Stk[TASK_STK_SIZE]; /* Task #3 task stack */
• OS_STK Task4Stk[TASK_STK_SIZE]; /* Task #4 task stack */
• OS_STK Task5Stk[TASK_STK_SIZE]; /* Task #5 task stack */

• OS_EVENT *AckMbox; /* Message mailboxes for Tasks #4 and #5 */
• OS_EVENT *TxMbox;

```

2 Mailboxes

Main()

```
• void main (void)
• {
•     OS_STK *ptos;
•     OS_STK *pbos;
•     INT32U size;

•     PC_DispClrScr(DISP_FGND_WHITE);          /* Clear the screen          */
•
•     OSInit();                                /* Initialize uC/OS-II        */
•
•     PC_DOSSaveReturn();                      /* Save environment to return to DOS */
•     PC_VectSet(uCOS, OSCtxSw);               /* Install uC/OS-II's context switch vector */
•
•     PC_ElapsedInit();                        /* Initialized elapsed time measurement */
•
•     ptos      = &TaskStartStk[TASK_STK_SIZE - 1]; /* TaskStart() will use Floating-Point */
•     pbos      = &TaskStartStk[0];
•     size      = TASK_STK_SIZE;
•     OSTaskStkInit\_FPE\_x86(&ptos, &pbos, &size);
•     OSTaskCreateExt(TaskStart,
•                   (void *)0,
•                   ptos,
•                   TASK_START_PRIO,
•                   TASK_START_ID,
•                   pbos,
•                   size,
•                   (void *)0,
•                   OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);
•
•     OSStart();                               /* Start multitasking        */
• }
```


TaskStart()

```

• void TaskStart (void *pdata)
• {
•   #if OS_CRITICAL_METHOD == 3                                /* Allocate storage for CPU status register */
•       OS_CPU_SR  cpu_sr;
•   #endif
•       INT16S      key;

•       pdata = pdata;                                          /* Prevent compiler warning */
•       TaskStartDispInit();                                    /* Setup the display */
•       OS_ENTER_CRITICAL();                                    /* Install uC/OS-II's clock tick ISR */
•       PC_VectSet(0x08, OSTickISR);
•       PC_SetTickRate(OS_TICKS_PER_SEC);                      /* Reprogram tick rate */
•       OS_EXIT_CRITICAL();

•       OSStatInit();                                           /* Initialize uC/OS-II's statistics */

•       AckMbox = OSMboxCreate((void *)0);                     /* Create 2 message mailboxes */
•       TxMbox  = OSMboxCreate((void *)0);

•       TaskStartCreateTasks();                                  /* Create all other tasks */

•       for (;;) {
•           TaskStartDisp();                                    /* Update the display */

•           if (PC_GetKey(&key)) {                               /* See if key has been pressed */
•               if (key == 0x1B) {                               /* Yes, see if it's the ESCAPE key */
•                   PC_DOSReturn();                             /* Yes, return to DOS */
•               }
•           }

•           OSCtxSwCtr = 0;                                       /* Clear context switch counter */
•           OSTimeDly(OS_TICKS_PER_SEC);                         /* Wait one second */
•       }
•   }

```

Create 2 mailboxes

The dummy loop wait for 'ESC'

Task1()

```
• void Task1 (void *pdata)
• {
•     INT8U      err;
•     OS_STK_DATA data;          /* Storage for task stack data */
•     INT16U      time;          /* Execution time (in uS) */
•     INT8U      i;
•     char        s[80];

•
•     pdata = pdata;
•     for (;;) {
•         for (i = 0; i < 7; i++) {
•             PC_ElapsedStart();
•             err = OSTaskStkChk(TASK_START_PRIO + i, &data);
•             time = PC_ElapsedStop();
•             if (err == OS_NO_ERR) {
•                 sprintf(s, "%4ld      %4ld      %4ld      %6d",
•                     data.OSFree + data.OSUsed,
•                     data.OSFree,
•                     data.OSUsed,
•                     time);
•                 PC_DispatchStr(19, 12 + i, s, DISP_FGND_BLACK + DISP_BGND_LIGHT_GRAY);
•             }
•         }
•         OSTimeDlyHMSM(0, 0, 0, 100);          /* Delay for 100 mS */
•     }
• }
```

Task4 and Task5

```
• void Task4 (void *data)
• {
•     char    txmsg;
•     INT8U   err;

•
•     data = data;
•     txmsg = 'A';
•     for (;;) {
•         OSMboxPost(TxMbox, (void *)&txmsg);           /* Send message to Task #5 */
•         OSMboxPend(AckMbox, 0, &err);                  /* Wait for acknowledgement from Task #5 */
•         txmsg++;                                         /* Next message to send */
•         if (txmsg == 'Z') {
•             txmsg = 'A';                               /* Start new series of messages */
•         }
•     }
• }

• void Task5 (void *data)
• {
•     char *rxmsg;
•     INT8U   err;

•
•     data = data;
•     for (;;) {
•         rxmsg = (char *)OSMboxPend(TxMbox, 0, &err);   /* Wait for message from Task #4 */
•         PC_Dispatch(70, 18, *rxmsg, DISP_FGND_YELLOW + DISP_BGND_BLUE);
•         OSTimeDlyHMSM(0, 0, 1, 0);                    /* Wait 1 second */
•         OSMboxPost(AckMbox, (void *)1);                /* Acknowledge reception of msg */
•     }
• }
```

MailBox

- A mailbox is a data exchange between tasks
 - A mailbox consists of a data pointer and a wait-list
- OSMboxPend():
 - The message in the mailbox is retrieved
 - If the mailbox is empty, the task is immediately **blocked** and moved to the wait-list
 - A time-out value can be specified
- OSMboxPost():
 - A message is deposited in the mailbox
 - If there is already a message in the mailbox, **an error is returned (not overwritten)**
 - If tasks waiting for a message from the mailbox, the task with the highest priority is removed from the wait-list and scheduled to run

OSTaskStkInit_FPE_x86()

- OSTaskStkInit_FPE_x86(&ptos, &pbos, &size)
- Passing the original top address, bottom address, and size of the stack
- On return, the arguments are modified and some stack space are reserved for floating point library
 - For context switches



OSCreateTaskExt()

- OSTaskCreateExt(
TaskStart,
(void *)0,
ptos,
TASK_START_PRIO,
TASK_START_ID,
pbos,
size,
(void *)0,
OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR
);

OSTaskStkCheck()

- Check for stack overflow
 - Criteria
 - $\text{bos} < (\text{tos} - \text{stack length})$
 - Who uses stacks?
 - Local variables,
 - arguments for procedure calls,
 - temporary storage for ISR's
 - When stacks are checked?
 - When a task is created
 - When OSTaskStkCheck() is called
 - No automatic stack checking

Summary: Example2

- Local variable, function calls, and ISR's will utilize the stack space of user tasks
 - ISR will use the stack of the task being interrupted
- If floating-point operations are needed, some stack space should be reserved
- Mailbox can be used to synchronize among tasks

OS_CPU.C – OSTaskStkInit_FPE_x86()

```
OS_STK Task1Stk[1000];
OS_STK Task2Stk[1000];

void main(void) {
    OS_STK *ptos;
    OS_STK *pbos;
    OS_Init();

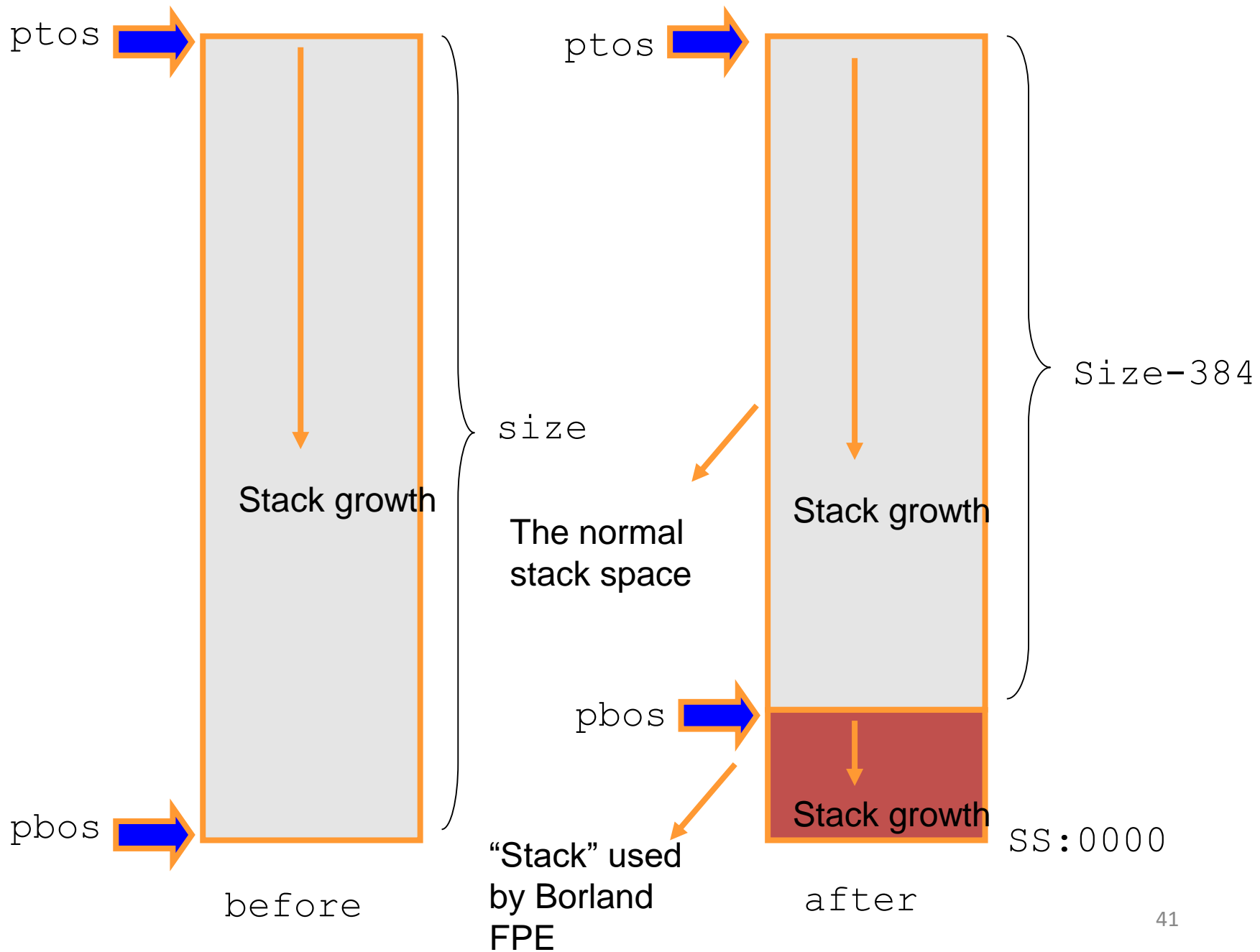
    ptos = &Task1Stk[999];
    pbos = &Task1Stk[0];
    size = 1000;

    OSTaskStkInit_FPE_x86(&ptos, &pbos, &size);
    OSTaskCreate(Task1, null, pbos, 10);

    ptos = &Task2Stk[999];
    pbos = &Task2Stk[0];
    size = 1000;

    OSTaskStkInit_FPE_x86(&ptos, &pbos, &size);
    OSTaskCreate(Task2, null, pbos,
                  11, 11, pbos, size, null, OSTask_OPT_SAVE_FP);

    OSStart();
}
```



```

void OSTaskStkInit_FPE_x86 (OS_STK **pptos, OS_STK **ppbos, INT32U *psize)
{
    /* 'Linear' version of top-of-stack      address      */
    INT32U  lin_tos;
    /* 'Linear' version of bottom-of-stack address      */
    INT32U  lin_bos;
    INT16U  seg;
    INT16U  off;
    INT32U  bytes;

    /* Decompose top-of-stack pointer into seg:off      */
    seg      = FP_SEG(*pptos);
    off      = FP_OFF(*pptos);
    /* Convert seg:off to linear address                */
    lin_tos  = ((INT32U)seg << 4) + (INT32U)off;
    /* Determine how many bytes for the stack          */
    bytes    = *psize * sizeof(OS_STK);
    /* Ensure paragraph alignment for BOS              */
    lin_bos  = (lin_tos - bytes + 15) & 0xFFFFFFF0L;

    /* Get new 'normalized' segment                    */
    seg      = (INT16U)(lin_bos >> 4);
    /* Create 'normalized' BOS pointer                 */
    *ppbos   = (OS_STK *)MK_FP(seg, 0x0000);
    /* Copy FP emulation memory to task's stack        */
    memcpy(*ppbos, MK_FP(_SS, 0), 384);
    /* Loose 16 bytes because of alignment             */
    bytes    = bytes - 16;
    /* Determine new top-of-stack                      */
    *pptos   = (OS_STK *)MK_FP(seg, (INT16U)bytes);
    /* Determine new bottom-of-stack                  */
    *ppbos   = (OS_STK *)MK_FP(seg, 384);
    bytes    = bytes - 384;
    /* Determine new stack size                       */
    *psize   = bytes / sizeof(OS_STK);
}

```

Remarks:
FP_OFF is a macro that can get or set the offset of the far pointer *p.
FP_SEG is a macro that gets or sets the segment value of the far pointer *p.
MK_FP is a macro that makes a far pointer from its component segment <seg> and offset <ofs> parts.

See you next class!