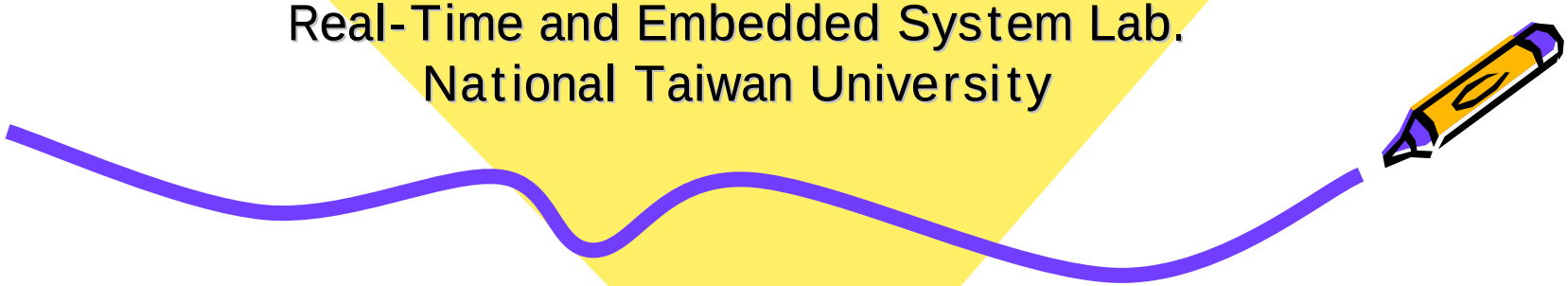
Chapter 1: Getting Started with uC/OS-II

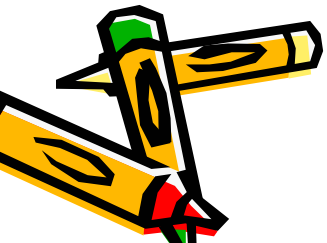
Dr. Li-Pin Chang
Real-Time and Embedded System Lab.
National Taiwan University



Text Book

- Jean J. Labresse, MicroC/OS-II: The Real-Time Kernel.

ISBN: 1-57820-103-9

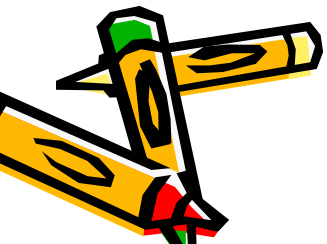


uC/OS-2

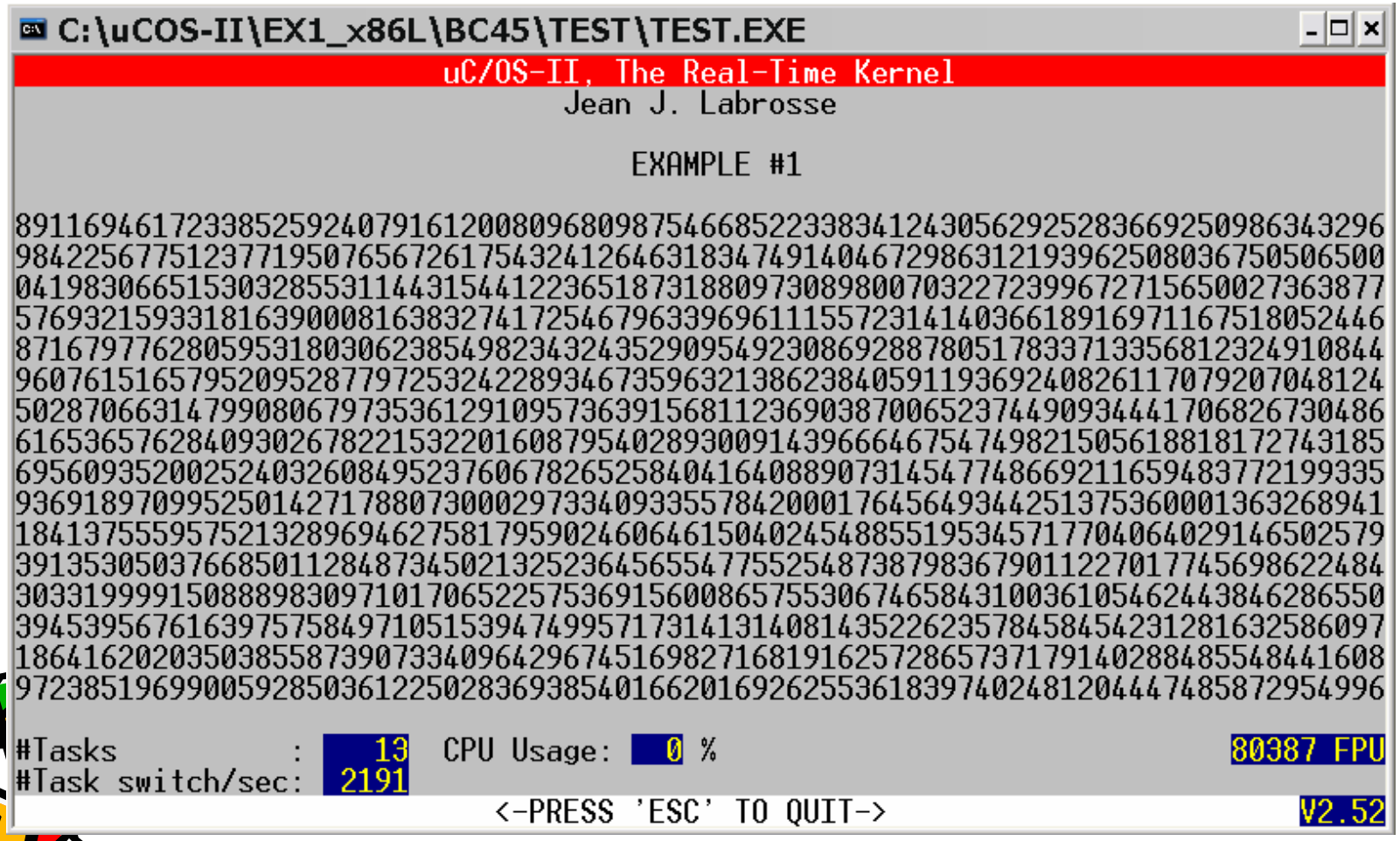
- A very small real-time kernel.
  - Memory footprint is about 20k for a fully functional kernel.
  - It's source is open.
  - Preemptible priority-driven real-time scheduling.
  - Support many platforms: x86, 68x, MIPS…
  - Very easy to develop: Borland C++ compiler and DOS (optional).
- However, it lacks of…
  - Resource synchronization protocols.
  - Sporadic task support.
  - Soft-real-time support.

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 real-time tasks?
  - How to use inter-task communication mechanism?
  - How to catch system event?

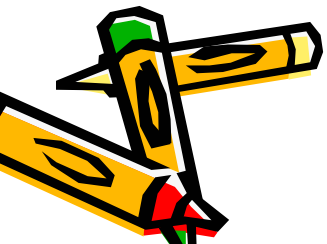


Example 1


C:\uCOS-II\EX1_x86L\BC45\TEST\TEST.EXE
uC/OS-II, The Real-Time Kernel
Jean J. Labrosse
EXAMPLE #1
8911694617233852592407916120080968098754668522338341243056292528366925098634329698422567751237719507656726175432412646318347491404672986312193962508036750506500041983066515303285531144315441223651873188097308980070322723996727156500273638775769321593318163900081638327417254679633969611155723141403661891697116751805244687167977628059531803062385498234324352909549230869288780517833713356812324910844960761516579520952877972532422893467359632138623840591193692408261170792070481245028706631479908067973536129109573639156811236903870065237449093444170682673048661653657628409302678221532201608795402893009143966646754749821505618818172743185695609352002524032608495237606782652584041640889073145477486692116594837721993359369189709952501427178807300029733409335578420001764564934425137536000136326894118413755595752132896946275817959024606461504024548855195345717704064029146502579391353050376685011284873450213252364565547755254873879836790112270177456986224843033199991508889830971017065225753691560086575530674658431003610546244384628655039453956761639757584971051539474995717314131408143522623578458454231281632586097186416202035038558739073340964296745169827168191625728657371791402884855484416089723851969900592850361225028369385401662016926255361839740248120447485872954996
#Tasks : 13 CPU Usage: 0 % 80387 FPU
#Task switch/sec: 2191
<-PRESS 'ESC' TO QUIT-> V2.52

Example 1

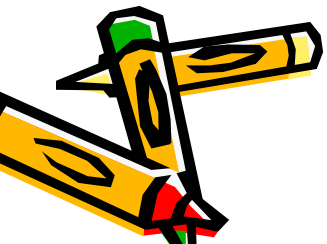
- uC/OS-2 can run with or without DOS.
  - DOS box under Windows
  - Boot directly from image on HDD/FDD.
  - Convenient to test and debug.



Example 1

- Files needed:
  - The main program (test.c)
  - The configuration for uC/OS-2(os_cfg.h)
  - The big include file (includes.h)

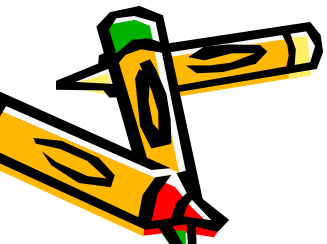
- Tools needed:
  - Borland C++ compiler (V3.1+)
  - A PC!!



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.



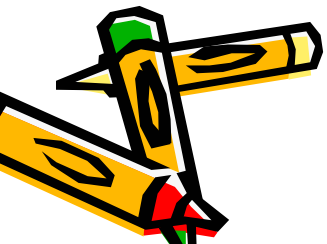
Example 1

```
#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 (explain later)

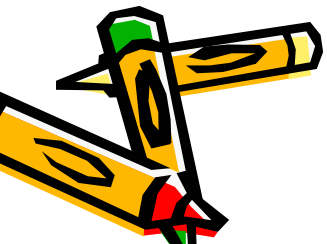
Main()

```
void main (void)
{
    PC_DispClrScr(DISP_FGND_WHITE + DISP_BGND_BLACK);          (1)
    OSInit();                                                  (2)
    PC_DOSSaveReturn();                                        (3)
    PC_VectSet(uCOS, OSCtxSw);                                 (4)
    RandomSem = OSSemCreate(1);                                (5)
    OSTaskCreate(TaskStart,                                    (6)
                 (void *)0,
                 (void *)&TaskStartStk[TASK_STK_SIZE-1],
                 0);
    OSStart();                                                 (7)
}
```

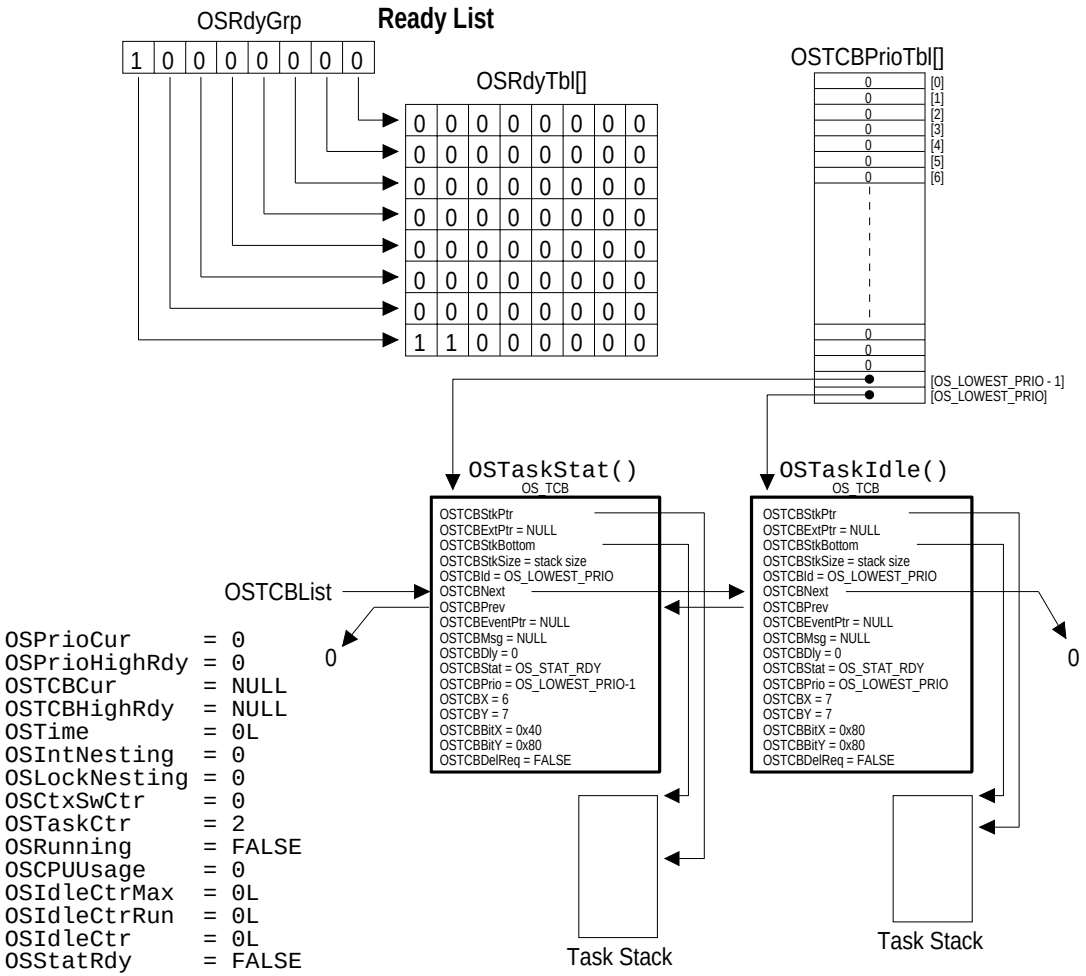


Main()

- 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.

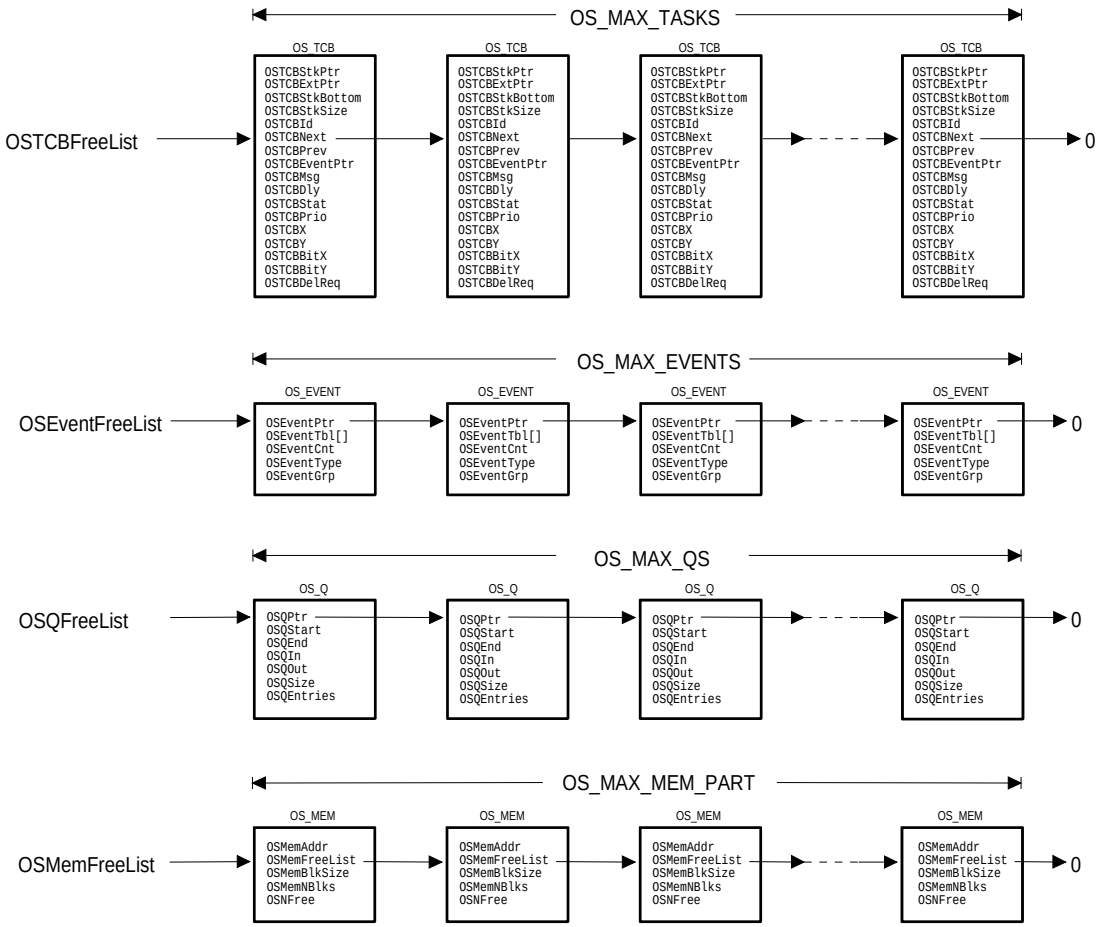


OSinit()



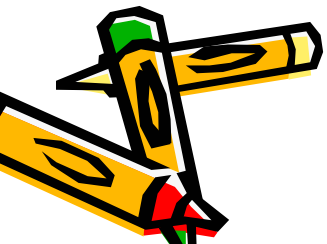

```
OSPrioCur     = 0
OSPrioHighRdy = 0
OSTCBCur      = NULL
OSTCBHighRdy  = NULL
OSTime        = 0L
OSIntNesting  = 0
OSLockNesting = 0
OSCtxSwCtr    = 0
OSTaskCtr     = 2
OSRunning     = FALSE
OSCPUUsage    = 0
OSIdleCtrMax  = 0L
OSIdleCtrRun  = 0L
OSIdleCtr     = 0L
OSStatRdy     = FALSE
```

OSinit()


OS_MAX_TASKS
OS_TCB
OSTCBStkPtr
OSTCBExtPtr
OSTCBStkBottom
OSTCBStkSize
OSTCBId
OSTCBNext
OSTCBPrev
OSTCBEventPtr
OSTCBMsg
OSTCBDly
OSTCBStat
OSTCBPrio
OSTCBX
OSTCBY
OSTCBBitX
OSTCBBitY
OSTCBDelReq
OSTCBFreeList
0
OS_MAX_EVENTS
OS_EVENT
OSEventPtr
OSEventTbl[]
OSEventCnt
OSEventType
OSEventGrp
OSEventFreeList
0
OS_MAX_QS
OS_Q
OSQPtr
OSQStart
OSQEnd
OSQIn
OSQOut
OSQSize
OSQEntries
OSQFreeList
0
OS_MAX_MEM_PART
OS_MEM
OSMemAddr
OSMemFreeList
OSMemBlkSize
OSMemNBlks
OSNFree
OSMemFreeList
0

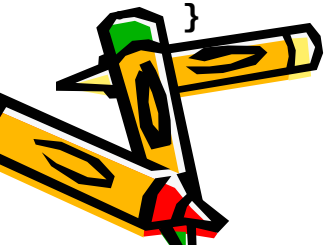
Main()

- PC_DOSSaveReturn()
  - Save the current status of DOS for the future restoration.
    - Interrupt vectors and the RTC tick rate.
  - Set a global returning point by calling setjump().
    - uC/OS-2 can come back here when terminating.
    - PC_DOSReturn()



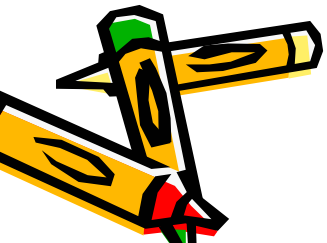
PC_DOSSaveReturn()

```
void PC_DOSSaveReturn (void)
{
    PC_ExitFlag  = FALSE;                                   (1)
    OSTickDOSCtr =     8;                                   (2)
    PC_TickISR   = PC_VectGet(VECT_TICK);                   (3)

    OS_ENTER_CRITICAL();
    PC_VectSet(VECT_DOS_CHAIN, PC_TickISR);                 (4)
    OS_EXIT_CRITICAL();

    setjmp(PC_JumpBuf);                                     (5)
    if (PC_ExitFlag == TRUE) {
        OS_ENTER_CRITICAL();
        PC_SetTickRate(18);                                 (6)
        PC_VectSet(VECT_TICK, PC_TickISR);                  (7)
        OS_EXIT_CRITICAL();
        PC_DispClrScr(DISP_FGND_WHITE + DISP_BGND_BLACK);   (8)
        exit(0);                                            (9)
    }
}
```



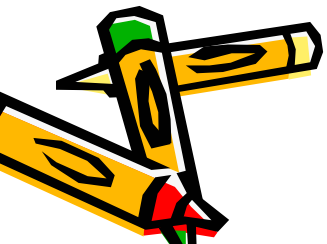
Main()

- PC_VectSet(uCOS,OSCtxSw)
  - Install the context switch handler.
  - Interrupt 0x08 under 80x86 family.
    - Invoked by int instruction.



Main()

- OSSemCreate()
  - Create a semaphore for resource synchronization.
    - To protect non-reentrant codes.
  - The created semaphore becomes a mutual exclusive mechanism if “1” is given as the initial value.
  - In this example, a semaphore is created to protect the standard C library “random()”.

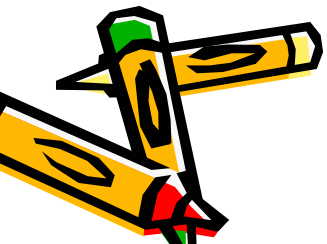


Main()

- OSTaskCreate()
  - Create tasks by the given arguments.
  - Tasks become “ready” after they are created.

- Task
  - An active entity which could do some computations.
  - **Priority, CPU registers, stack, text, housekeeping status**.
  - uC/OS-2 allows maximum 63 tasks created.

- The uC/OS-2 picks up the highest priority task to run on context-switch.
  - Tightly coupled with RTC ISR.



OSTaskCreate()

- **OSTaskCreate(**

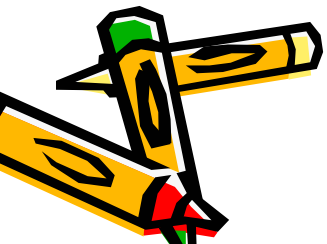
```
TaskStart,
(void *)0,
&TaskStartStk[TASK_STK_SIZE - 1],
 0
);
```

Entry point of the task (a pointer to function)

User-specified data

Priority (0=hightest)

Top of Stack



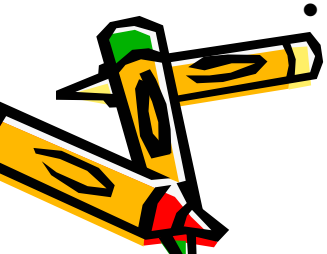
TaskStart()

```
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                          */
    }
}
```

Change the ticking rate

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()

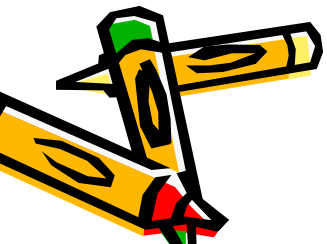
```
static  void  TaskStartCreateTasks (void)
{
    INT8U  i;


    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

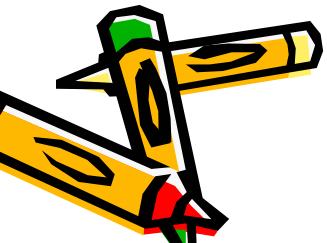
Stack



Task()

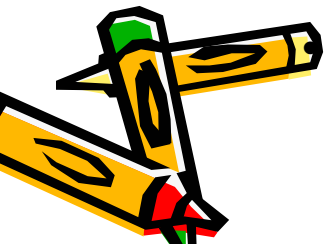
```
void  Task (void *pdata)
{
    INT8U  x;
    INT8U  y;
    INT8U  err;


    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_DispChar(x, y + 5, *(char *)pdata, DISP_FGND_BLACK + DISP_BGND_LIGHT_GRAY);
        OSTimeDly(1);                 /* Delay 1 clock tick                            */
    }
}
```

Semaphore operations.



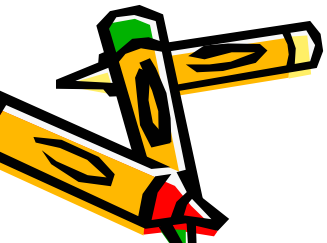
Semaphores

- 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.



Main()

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

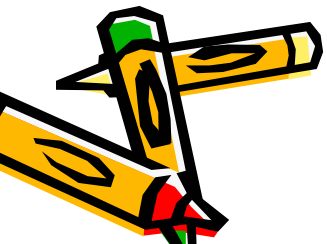


Summary: Example 1

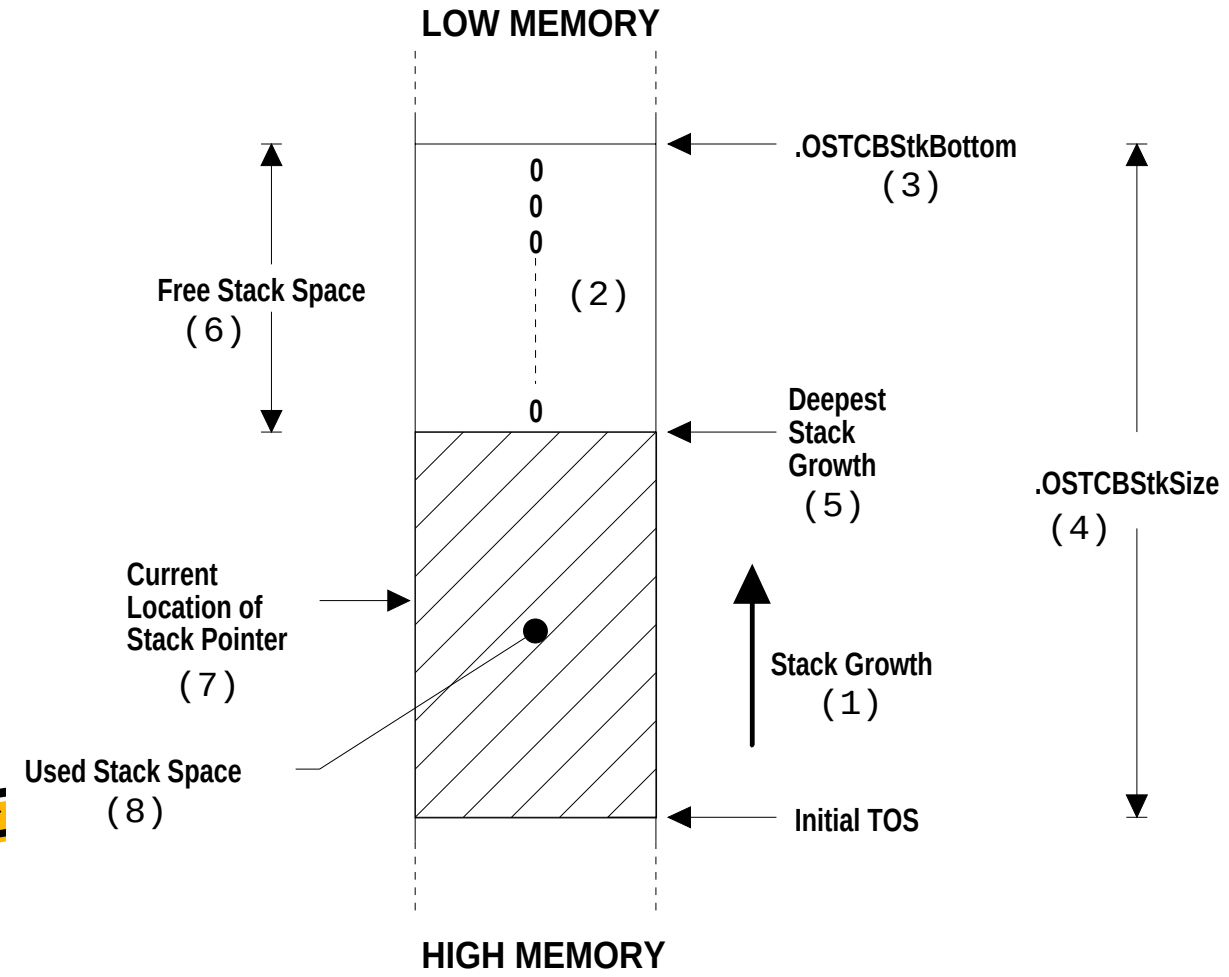
- uC/OS-2 is initialized and started by calling OSInit() and OSStart(), respectively.
- Before uC/OS-2 is started,
  - DOS status is saved by calling PC_DOSSaveReturn().
  - Context switch handler is installed by calling PC_VectSet().
  - User tasks must be created by OSTaskCreate().
- Shared resources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

Example 2

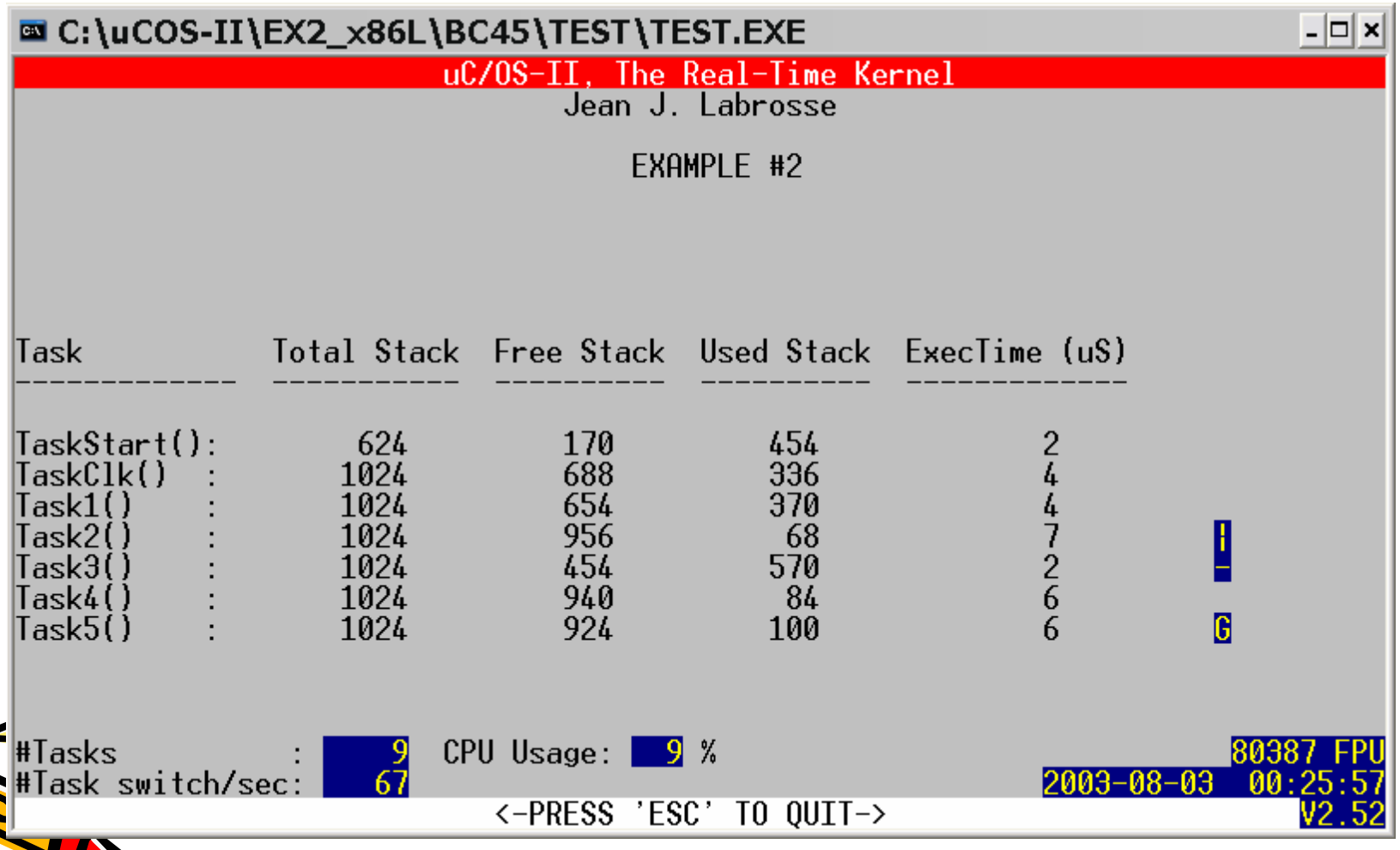
- Example 2 focuses on:
  - More task creation options.
  - **Stack usage** of each task.
  - **Floating point** operations.
  - Communication through **mailbox**.



Stack Usage of a Task

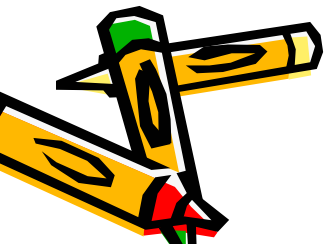


Example 2


C:\uCOS-II\EX2_x86L\BC45\TEST\TEST.EXE
uC/OS-II, The Real-Time Kernel
Jean J. Labrosse
EXAMPLE #2
Task Total Stack Free Stack Used Stack ExecTime (uS)
TaskStart(): 624 170 454 2
TaskClk() : 1024 688 336 4
Task1() : 1024 654 370 4
Task2() : 1024 956 68 7
Task3() : 1024 454 570 2
Task4() : 1024 940 84 6
Task5() : 1024 924 100 6
#Tasks : 9 CPU Usage: 9 %
#Task switch/sec: 67
80387 FPU
2003-08-03 00:25:57
<-PRESS 'ESC' TO QUIT->
V2.52

```
#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_DispStr(19, 12 + i, s, DISP_FGND_BLACK + DISP_BGND_LIGHT_GRAY);
            }
        }
        OSTimeDlyHMSM(0, 0, 0, 100);                    /* Delay for 100 mS           */
    }
}
```

errata

The local variables

Task1: total 1024 Free 654 Used 370

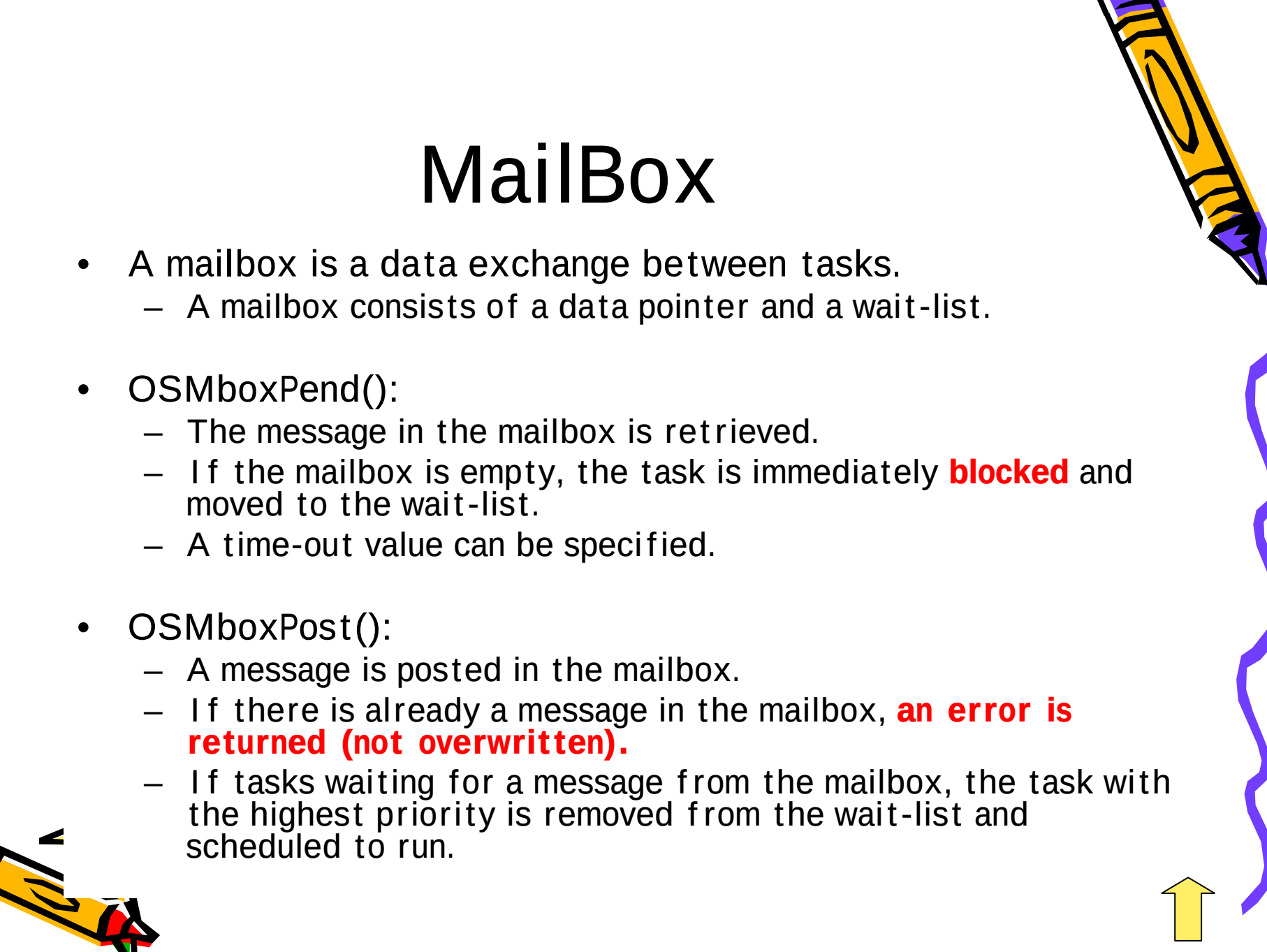
```
void  Task2 (void *data)
{
    data = data;
    for (;;) {
        PC_DispChar(70, 15, '|',  DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(10);
        PC_DispChar(70, 15, '/',  DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(10);
        PC_DispChar(70, 15, '-',  DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(10);
        PC_DispChar(70, 15, '\\', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(10);
    }
}

void  Task3 (void *data)
{
    char    dummy[500];
    INT16U  i;


    data = data;
    for (i = 0; i < 499; i++) {        /* Use up the stack with 'junk'                    */
        dummy[i] = '?';
    }
    for (;;) {
        PC_DispChar(70, 16, '|',  DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(20);
        PC_DispChar(70, 16, '\\', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(20);
        PC_DispChar(70, 16, '-',  DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(20);
        PC_DispChar(70, 16, '/',  DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(20);
    }
}
```

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_DispChar(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  */
    }
}
```

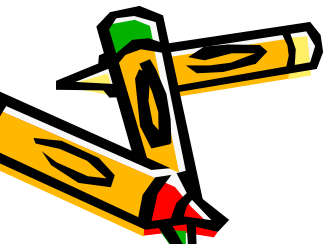
MailBox

- 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 posted 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.



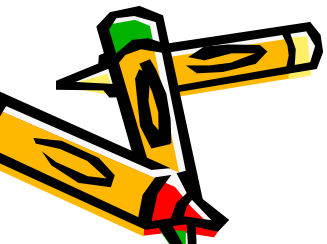
OSTaskStkInit_FPE_x86()

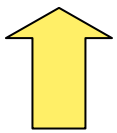
- OSTaskStkInit_FPE_x86(&ptos, &pbos, &size)
- Passing the original top address, bottom address, and size of the stack.
- On return, the arguments are modified and some stack space are reserved for floating point library.
  - For context switches.



OSCreateTaskExt()

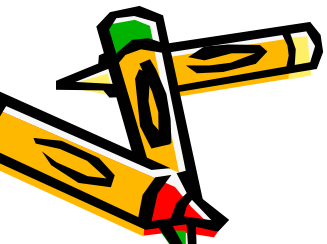
- OSTaskCreateExt(
  TaskStart,
  (void *)0,
  ptos,
  TASK_START_PRIO,
  TASK_START_ID,
  pbos,
  size,
  (void *)0,
  OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR
  );





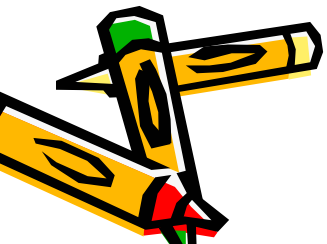
OSTaskStkCheck()

- Check for stack overflow.
  - bos < (tos – stack length)
  - Local variables, arguments for procedure calls, temporary storage for ISR's.
  - uC/OS-2 can check for stack overflow on the creation of tasks and when OSTaskStkCheck() is called.
  - uC/OS-2 does not automatically check stacks.



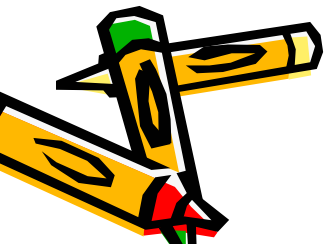
Summary: Example2

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

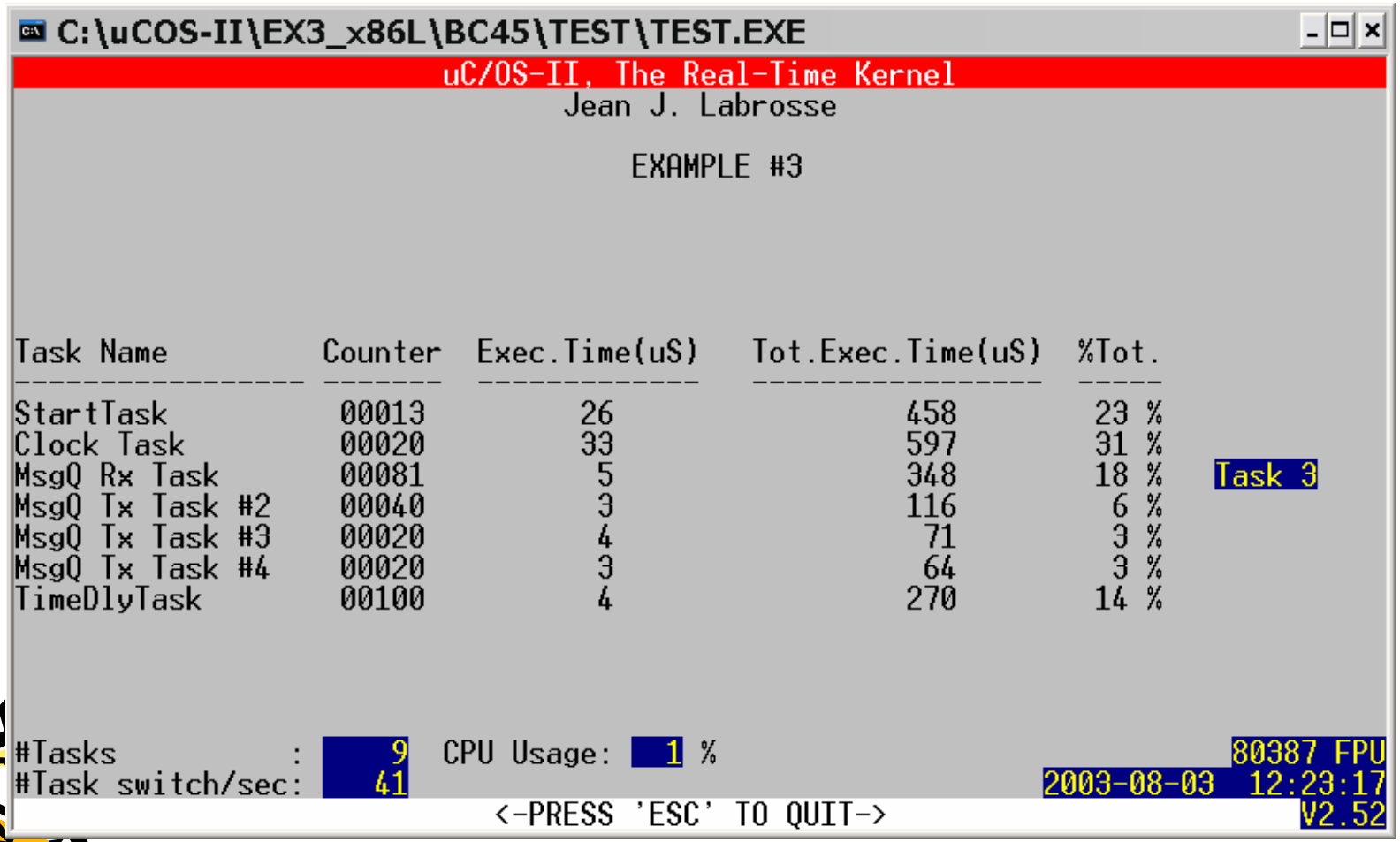


Example 3

- Passing user-specified data structures on task creations.
- Using message queues.
- Demonstrating how to use OS hooks to receive desired event from the uC/OS-2.



Example 3


C:\uCOS-II\EX3_x86L\BC45\TEST\TEST.EXE
uC/OS-II, The Real-Time Kernel
Jean J. Labrosse
EXAMPLE #3
Task Name Counter Exec.Time(uS) Tot.Exec.Time(uS) %Tot.
StartTask 00013 26 458 23 %
Clock Task 00020 33 597 31 %
MsgQ Rx Task 00081 5 348 18 %
MsgQ Tx Task #2 00040 3 116 6 %
MsgQ Tx Task #3 00020 4 71 3 %
MsgQ Tx Task #4 00020 3 64 3 %
TimeDlyTask 00100 4 270 14 %
Task 3
#Tasks : 9
#Task switch/sec: 41
CPU Usage: 1 %
80387 FPU
2003-08-03 12:23:17
V2.52
<-PRESS 'ESC' TO QUIT->

```
#define          TASK_STK_SIZE     512              /* Size of each task's stacks (# of WORDs)       */

#define          TASK_START_ID       0              /* Application tasks                             */
#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

#define          MSG_QUEUE_SIZE     20              /* Size of message queue used in example         */

typedef struct {
    char    TaskName[30];
    INT16U  TaskCtr;
    INT16U  TaskExecTime;
    INT32U  TaskTotExecTime;
} TASK_USER_DATA;


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                         */

TASK_USER_DATA  TaskUserData[7];

OS_EVENT       *MsgQueue;                           /* Message queue pointer                         */
void           *MsgQueueTbl[20];                    /* Storage for messages                          */
```

User-defined data structure to pass to tasks

Message queue and an array of event

```
void  Task1 (void *pdata)
{
   char  *msg;
   INT8U  err;


   pdata = pdata;
   for (;;) {
      msg = (char *)OSQPend(MsgQueue, 0, &err);
      PC_DispStr(70, 13, msg, DISP_FGND_YELLOW + DISP_BGND_BLUE);
      OSTimeDlyHMSM(0, 0, 0, 100);
   }
}


void  Task2 (void *pdata)
{
   char  msg[20];


   pdata = pdata;
   strcpy(&msg[0], "Task 2");
   for (;;) {
      OSQPost(MsgQueue, (void *)&msg[0]);
      OSTimeDlyHMSM(0, 0, 0, 500);
   }
}
```

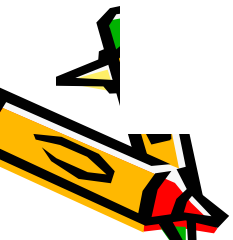
Task 2, 3, 4 are functionally identical.

Message Queues

- A message queue consists an array of elements and a wait-list.
- Different from a mailbox, a message queue can hold many data elements (in a FIFO basis).
- As same as mailboxes, there can be multiple tasks pend/post to a message queue.
- **OSQPost**(): a message is appended to the queue. The highest-priority pending task (in the wait-list) receives the message and is scheduled to run, if any.
- **OSQPend**(): a message is removed from the array of elements. If no message can be retrieved, the task is moved to the wait-list and becomes blocked.

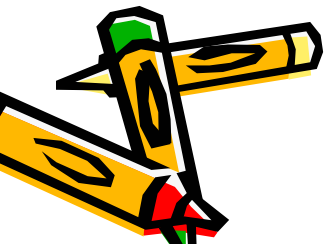
Hooks

- A hook function will be called by uC/OS-2 when the corresponding event occurs.
  - Event handlers in user programs.
  - For example, OSTaskSwHook () is called every time when context switch occurs.
- The hooks are specified in compile time in uC/OS-2.
  - Since it is an embedded OS.
  - Most OS's can register and un-register hooks.



User Customizable Hooks

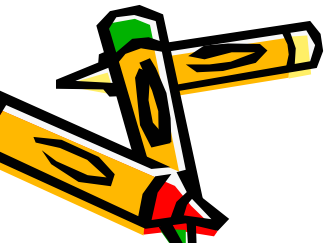
- void OSInitHookBegin (void)
- void OSInitHookEnd (void)
- void OSTaskCreateHook (OS_TCB *ptcb)
- void OSTaskDelHook (OS_TCB *ptcb)
- void OSTaskIdleHook (void)
- void OSTaskStatHook (void)
- void OSTaskSwHook (void)
- void OSTCBInitHook (OS_TCB *ptcb)
- void OSTimeTickHook (void)



```
void  OSTaskStatHook (void)
{
    char    s[80];
    INT8U   i;
    INT32U  total;
    INT8U   pct;


    total = 0L;                                          /* Totalize TOT. EXEC. TIME for each task */
    for (i = 0; i < 7; i++) {
        total += TaskUserData[i].TaskTotExecTime;
        DispTaskStat(i);                                 /* Display task data                      */
    }
    if (total > 0) {
        for (i = 0; i < 7; i++) {                        /* Derive percentage of each task         */
            pct = 100 * TaskUserData[i].TaskTotExecTime / total;
            sprintf(s, "%3d %%", pct);
            PC_DispStr(62, i + 11, s, DISP_FGND_BLACK + DISP_BGND_LIGHT_GRAY);
        }
    }
    if (total > 1000000000L) {                           /* Reset total time counters at 1 billion */
        for (i = 0; i < 7; i++) {
            TaskUserData[i].TaskTotExecTime = 0L;
        }
    }
}

void  OSTaskSwHook (void)
{
    INT16U           time;
    TASK_USER_DATA  *puser;


    time  = PC_ElapsedStop();                    /* This task is done                                */
    PC_ElapsedStart();                           /* Start for next task                              */
    puser = OSTCBCur->OSTCBExtPtr;               /* Point to used data                               */
    if (puser != (TASK_USER_DATA *)0) {
        puser->TaskCtr++;                        /* Increment task counter                           */
        puser->TaskExecTime     = time;          /* Update the task's execution time                 */
        puser->TaskTotExecTime += time;          /* Update the task's total execution time           */
    }
}
```

Summary: Example 3

- Message queues can be used to synchronize among tasks.
  - Multiple message can be held in the queue.
  - Multiple tasks can pend/post to message queues simultaneously.
- Hooks can be used to do some user-specific computations on certain OS events occurs.
  - They are specified in compile time.



Summary: Getting Started with uC/OS-2

- Have you understood:
  - how to write a dummy uC/OS-2 program?
  - how the control flows among procedures?
  - how tasks are created?
  - how tasks are synchronized by semaphore, mailbox, and message queues?
  - how the space of stacks are utilized?
  - how to capture system events?