Embedded Operating Systems- Final Project

Che-Wei Chang

chewei@mail.cgu.edu.tw

Department of Computer Science and Information Engineering, Chang Gung University

Report

- Only four A4 pages
- 12 pt words
- Deadline is 23:59 2020/01/03
- File name: EOS-Project-StudentID.zip
- Required Files: The source code files and the report
- In the report, remember to provide your names, student IDs, and group ID.
- Send it to my email: chewei@mail.cgu.edu.tw
- Email title: EOS Project StudentID

The Requirements of Final Presentation

- Presentation is only for 10 minutes
  - Quickly go through the implementation
  - Talk more about the problems you solved
  - Highlight your extra exercise
- Live demo is required
  - Bring your source code
- I will ask each of you a question

The µC/OS-II File Structure

Application Code (Your Code!)

Processor Independent Implementations

- Scheduling policy
- Event flags
- Semaphores
- Mailboxes
- Event queues
- Task management
- Time management
- Memory management

Application Specific Configurations

- OS_CFG.H
- Max # of tasks
- Max Queue length
- …

uC/OS-II Port for Processor Specific Codes

Software

Hardware

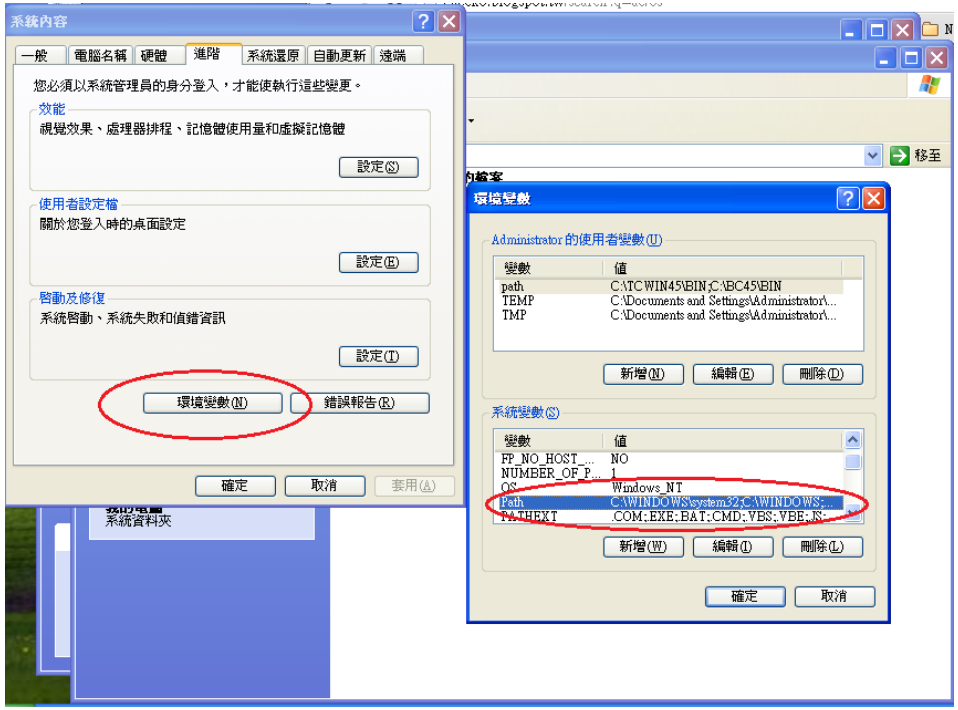
CPU | Timer

Requirements of µC/OS-II Emulator

- Operating System
  - Windows XP 32bits
  - Use virtual machine to install the OS
  - Install “Guest Additions” for Virtualbox
- Tools
  - Borland C++ compiler (V4.5)
    - BC45 is the compiler
  - Turbo Assembler
    - The assembler is in tasm
  - The source code and the emulation environment of µC/OS-II
    - SOFTWARE is the package
- Full Package
  - Download it from the course website with password: csie2018

Borland C++ Compiler

- Download Borland C++ and install it on your windows XP environment
  - Double click the "INSTALL.EXE"
- Add ";C:\BC45\BIN" to your system Path



Turbo Assembler

- Download Turbo assembler and unzip the file
- Copy "\tasm\BIN\TASM.EXE" to your "C:\BC45\BIN"
  - Include the missing assembler which is going to be used during we compile the source code of μC/OS-II

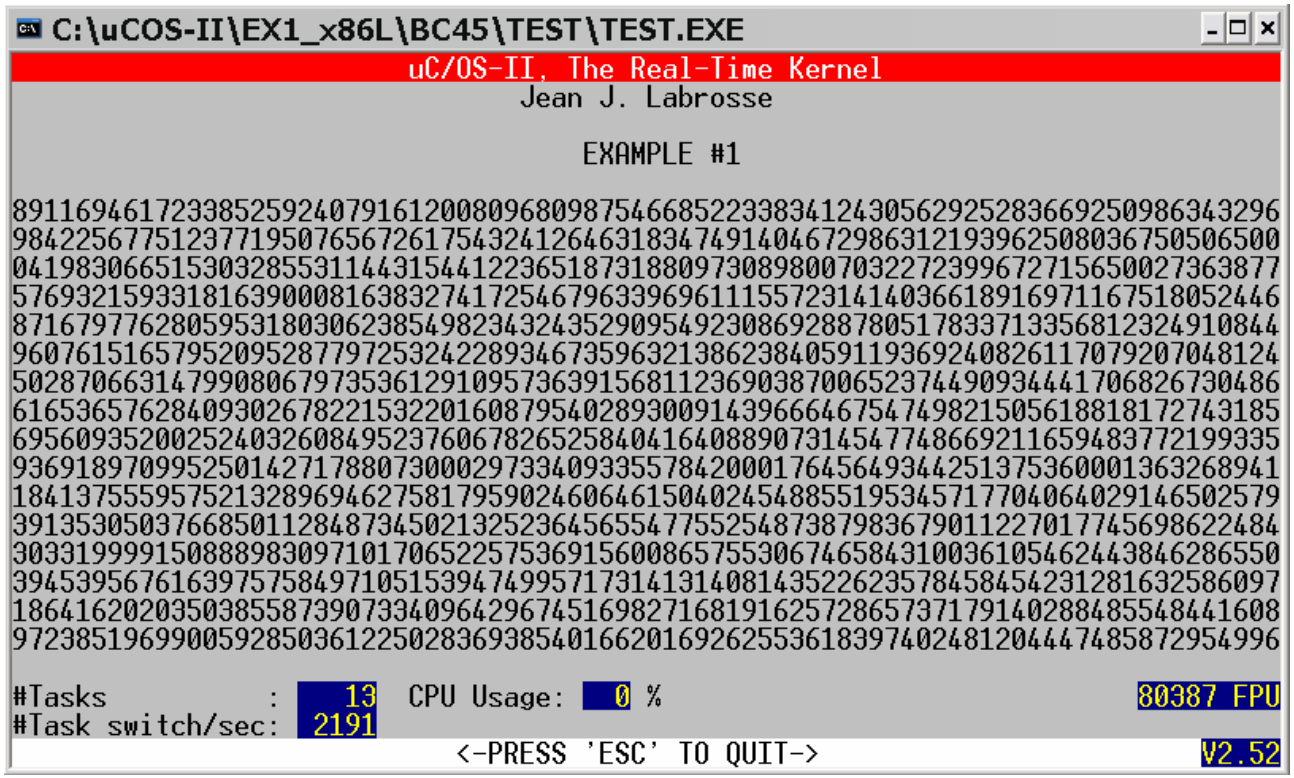
Compile µC/OS-II Example Code

- Download the source code and emulator µC/OS-II
  - It is recommended to put the source code package "SOFTWARE" directly in C:\
- Test the first example
  - Execute C:\SOFTWARE\uCOS-II\EX1_x86L\BC45\TEST\TEST.EXE
  - Press ECS to leave
- Rename or remove the executable file
  - Rename TEST.EXE
- Compile the µC/OS-II and the source code of the first example
  - Run C:\SOFTWARE\uCOS-II\EX1_x86L\BC45\TEST\MAKETEST.BAT
  - A new "TEST.EXE" will be created if we compile it successfully

Common Mistakes

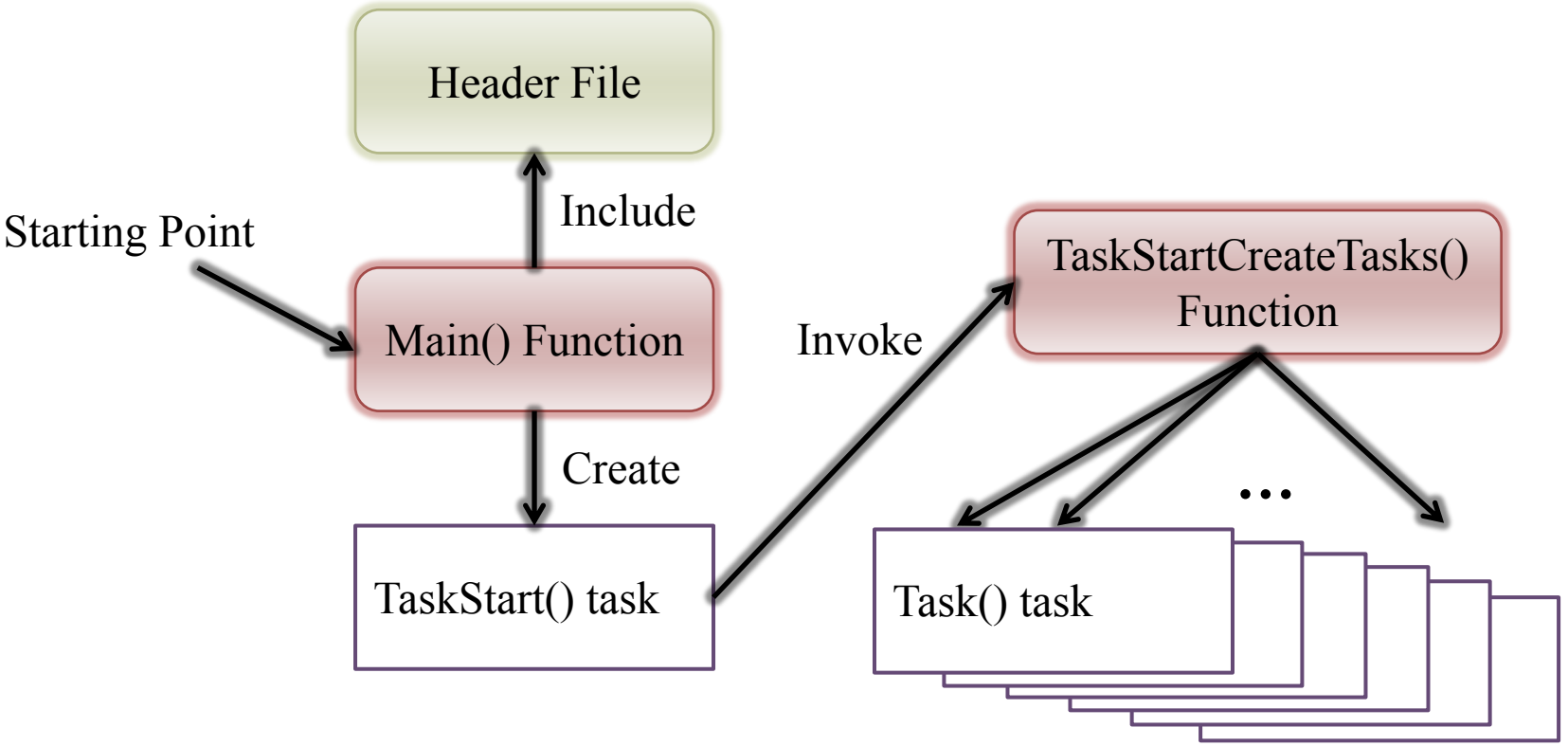
- Did you directly put the package “SOFTWARE” in C:\ ?
- Have you copied the correct file “TASM.EXE” to your “C:\BC45\BIN” directory?
- Did you set the Path correctly?
  - See the picture in Page 6
  - There is no space

An Example on µC/OS-II: Multitasking



- Three system tasks
- Ten application tasks randomly prints its number

Multitasking: Workflow



Multitasking: TEST.C

(\SOFTWARE\uCOS-II\EX1_x86L\BC45\SOURCE\TEST.C)

```
#include "includes.h"
/*
*********************************************************************************
CONSTANTS
*********************************************************************************
*/
#define TASK_STK_SIZE 512
#define N_TASKS 10
/*
*********************************************************************************
VARIABLES
*********************************************************************************
*/
OS_STK TaskStk[N_TASKS][TASK_STK_SIZE];
OS_STK TaskStartStk[TASK_STK_SIZE];
char TaskData[N_TASKS];
OS_EVENT *RandomSem;
```


Multitasking: Main()

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

Entry point of the task (a pointer to a function)

User-specified data

Top of stack

Priority (0=hightest)

Multitasking: TaskStart()

```
void TaskStart (void *pdata)
{
        /*skip the details of setting*/
        OSStatInit();
        TaskStartCreateTasks();
        for (;;)
        {
                if (PC_GetKey(&key) == TRUE)
                {
                        if (key == 0x1B) { PC_DOSReturn(); }
                }
                OSTimeDlyHMSM(0, 0, 1, 0);
        }
}
```

Call the function to create the other tasks

See if the ESCAPE key has been pressed

Wait one second

Multitasking: 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 task (a pointer to function)

Argument: character to print

Top of stack

Priority

Multitasking: 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 */
            PC_DispChar(x, y + 5, *(char *)pdata, DISP_FGND_BLACK +DISP_BGND_LIGHT_GRAY);
            /* Display the task number on the screen */
            OSTimeDly(1);
            /* Delay 1 clock tick */
        }
}
```

Randomly pick up the position to print its data

Print & delay

OSinit()

(\SOFTWARE\uCOS-II\SOURCE\OS_CORE.C)

- Initialize the internal structures of μC/OS-II and MUST be called before any services
- Internal structures of μC/OS-2
  - Task ready list
  - Priority table
  - Task control blocks (TCB)
  - Free pool
- Create housekeeping tasks
  - The idle task
  - The statistics task

PC_DOSSaveReturn()
(\SOFTWARE\BLOCKS\PC\BC45\PC.C)

- 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()
  - µC/OS-II can come back here when it terminates.
  - PC_DOSReturn()

PC_VectSet(uCOS,OSCtxSw)
(\SOFTWARE\BLOCKS\PC\BC45\PC.C)

- Install the context switch handler
- Interrupt 0x08 (timer) under 80x86 family
  - Invoked by INT instruction

OSStart()
(SOFTWARE\uCOS-II\EX1_x86L\BC45\SOURCE\CORE.C)

- Start multitasking of μC/OS-II
- It never returns to main()
- μC/OS-II is terminated if PC_DOSReturn() is called

Final Project 1

Implement EDF Scheduling

- Task Scheduling
  - The priority ceiling of a semaphore is the priority of the highest priority task that may lock the semaphore
  - The Basic Priority Inheritance Protocol + Priority Ceiling
  - A task J may successfully lock a semaphore S if S is available, and the priority of J is higher than the highest priority ceiling of all semaphores currently locked by tasks other than J
    - See OS_Sched( ) for scheduling policy
    - See OSTimeTick() for time management
    - See OSIntExit( ) for the interrupt management
    - See OSTaskChangePrio() for changing the priority of a task
- Provide the RM/EDF Scheduler
  - Input: A task set, each task is with its execution time and period
  - Output: The printed result of each task

Input

- The input format should be as follows
  - Your program should have the capability to create the assigned number of tasks and their corresponding period and execution time.
  - Example: taskset.txt

    3 //number of task

    1 3 // task 1: (execution time 1, period 1)

    2 9 // task 2: (execution time 2, period 2)

    4 12 // task 3: (execution time 3, period 3)
- The number of tasks is no more than 7

Input Example

```
4
1 12
1 7
2 19
3 20
```

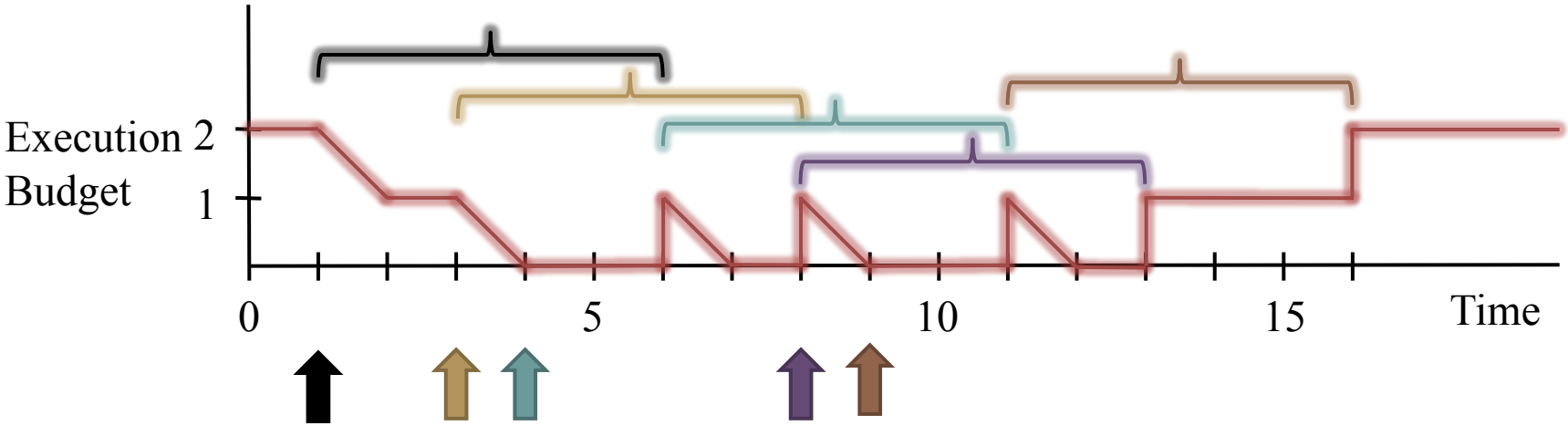

Output

- Your program output must show the following information
  - A sequence of the running task over time
  - The time when context switch occurred
- A report to describe your implementation
  - Relationship of each function
  - Implementation flow chart
  - Implementation details

Final Project 2

Implement Sporadic Server

- A sporadic server has a replenishment period 5 and an execution budget 2
- Each event consumes the execution 1
- Events arrive at 1, 3, 4, 8, 9



Input

- The input format should be as follows
  - Your program should have the capability to create the assigned number of tasks and their corresponding period and execution time
  - Assume that at starting time 0, the system has full execution budget
  - Example: taskset.txt

    4 6 5// execution budget: 4 replenishment period: 6 number of events: 5

    2 2 5 1 10 1 14 2 15 2 /* the first event arrives a time 2 with execution time 2, the second event arrive at time 5 with execution time 1, and so forth */
- The number of events is no more than 20
- The arrival time of the last even is no late than 100
- The execution budget is no more than 6
- The replenishment period is no longer than 10

Input Example

```
2 5 5
1 1 3 1 4 1 8 1 9 1
```


Output

- Your program output must show the following information
  - A sequence of the running task over time
  - You can not just draw the results, there should be some tasks running
- A report to describe your implementation
  - Relationship of each function
  - Implementation flow chart
  - Implementation details

Final Project 3

Implement Priority Ceiling Protocol

- PCP Rules
  - The priority ceiling of a semaphore is the priority of the highest priority task that may lock the semaphore
  - The Basic Priority Inheritance Protocol + Priority Ceiling
  - A task *J* may successfully lock a semaphore S if S is available, and the priority of *J* is higher than the highest priority ceiling of all semaphores currently locked by tasks other than *J*
- OS Modification
  - You have to read Chapter 7 Semaphore Management
  - See OSSemPend() and OSSemPost()
  - Keep another tables to record the priority ceiling and the state of each semaphore

Input

- The input format should be as follows
  - Your program should have the capability to create the assigned number of tasks and their corresponding period and execution time.
  - Example: taskset.txt

    3 4 // 3 tasks (tasks 1, 2, 3) and 4 semaphores (semaphores 1, 2, 3, 4)

    2 6 1 4 0 1/* task 1: (execution time 2, period 6, use 1 semaphore, semaphore is used at time 0 during its execution for time duration 1) */

    2 9 2 3 0 1 2 1 1

    4 12 2 2 0 2 1 1 1
- The number of tasks is no more than 7
- The number of semaphores is no more than 7
- You can use RM for the scheduling algorithm

Input Example

```
4 5
3 15 0
3 20 2 1 0 2 2 1 1
2 25 2 2 0 1 3 1 1
3 30 3 3 0 1 4 0 1 5 2 1
```


Output

- Your program output must show the following information
  - A sequence of the running task over time
  - The time when context switch occurred
- A report to describe your implementation
  - Relationship of each function
  - Implementation flow chart
  - Implementation details

Implementation Hints

EDF Scheduler

- Based on the RM scheduler
  - Do the RM shorting at the beginning
- Whenever a task is complete
  - Do not need to change anything
- Whenever a task ready
  - /SOFTWARE/uCOS-II/SOURCE/OS_TIME.C
    - Read OSTimeDly() for how to use OSTCBDly
  - /SOFTWARE/uCOS-II/SOURCE/OS_CORE.C
    - Read OSTimeTick () for how to wake up a task
    - When a task is ready, check its deadline with the current tasks
    - Search OSTaskChangePrio in the textbook

Sporadic Server

- Based on the RM scheduler implementation
  - The period can be very long
  - Create all task at the beginning, but make them sleep utile they arrive
- Execution Budget
  - Keep a global valuable for the execution budget
  - Keep a timestamp whenever a task starts to use computing resource
  - In /SOFTWARE/uCOS-II/SOURCE/OS_CORE.C: OSTimeTick (),
    - Decrease the budget when doing computing
    - Replenish the budget when it should be done
- Whenever the budget is not enough
  - OSTaskResume() another very high-priority task
  - OSTaskSuspend() the high-priority task

Priority Ceiling Protocol

- Based on the RM scheduler implementation
  - Use RM scheduling in this project
- Read the input and derive the priority ceiling of each semaphore
- Modify the code of a task
  - Keep a time point list a semaphore list for semaphore locking
  - Call a PCP semaphore locking function which is provided by yourself
  - After it lock the semaphore, keep a counter for releasing the semaphore by a PCP semaphore releasing function which is provided by yourself
- PCP_Lock function implementation
  - When a task should blocked by PCP
    - Call OSTaskSuspend() and note the priority of the task
  - When a task can lock the the semaphore by OSSemPend()
    - Update the list of currently used semaphores
- PCP_Release function implementation
  - Update the list of currently used semaphores
  - Call OSSemPost() to release the semaphore
  - Call OSTaskResume() if a task is no more blocked by PCP