Operating System Concepts

Che-Wei Chang

chewei@mail.cgu.edu.tw

Department of Computer Science and Information Engineering, Chang Gung University

Final Project–
An Real-Time OS: µC/OS-II
Quick Overview

Introduction of µC/OS-II (1/2)

- The name is from micro-controller operating system, version 2
- µC/OS-II is certified in an avionics product by FAA in July 2000 and is also used in the Mars Curiosity Rover
- It is a very small real-time kernel
  - Memory footprint is about 20KB for a fully functional kernel
  - Source code is about 5,500 lines, mostly in ANSI C
  - It's source is open but not free for commercial usages
- Preemptible priority-driven real-time scheduling
  - 64 priority levels (max 64 tasks)
  - 8 reserved for µC/OS-II
  - Each task is an infinite loop

Introduction of µC/OS-II (2/2)

- Deterministic execution times for most µC/OS-II functions and services
- Nested interrupts could go up to 256 levels
- Supports of various 8-bit to 64-bit platforms: x86, ARM, MIPS, 8051, etc.
- Easy for development: Borland C++ compiler and DOS (optional)
- However, uC/OS-II still lacks of the following features:
  - Resource synchronization protocol
  - Soft-real-time support

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

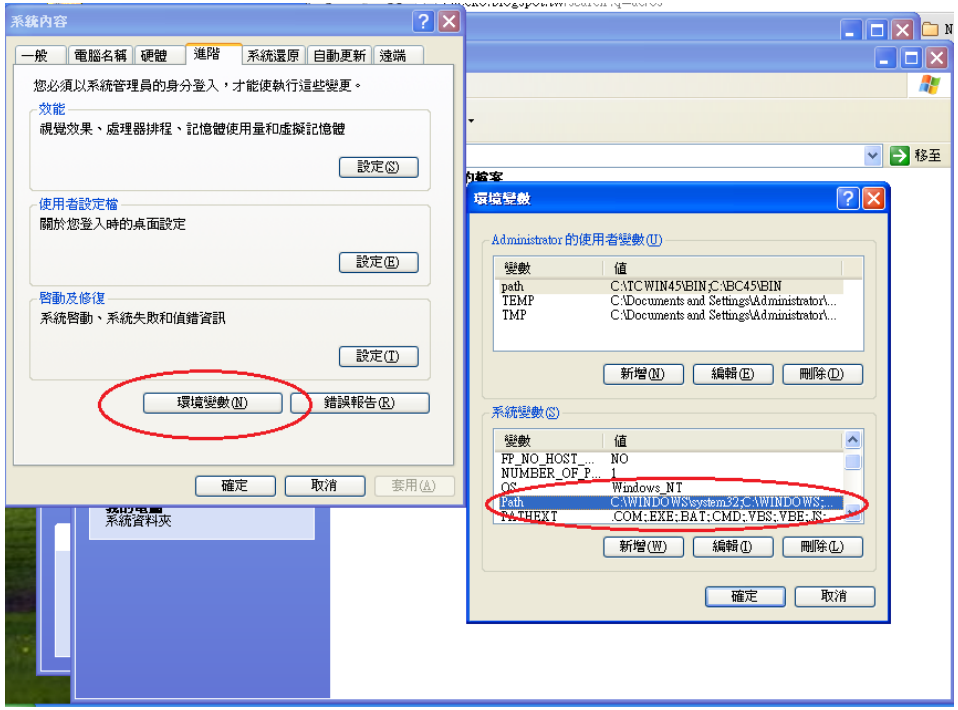
CPU | Timer

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: csie2020
  - https://www.csie.cgu.edu.tw/~chewei/files/ucOSII_ProjectPackage.zip
  - https://www.csie.cgu.edu.tw/~chewei/files/Files.zip

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

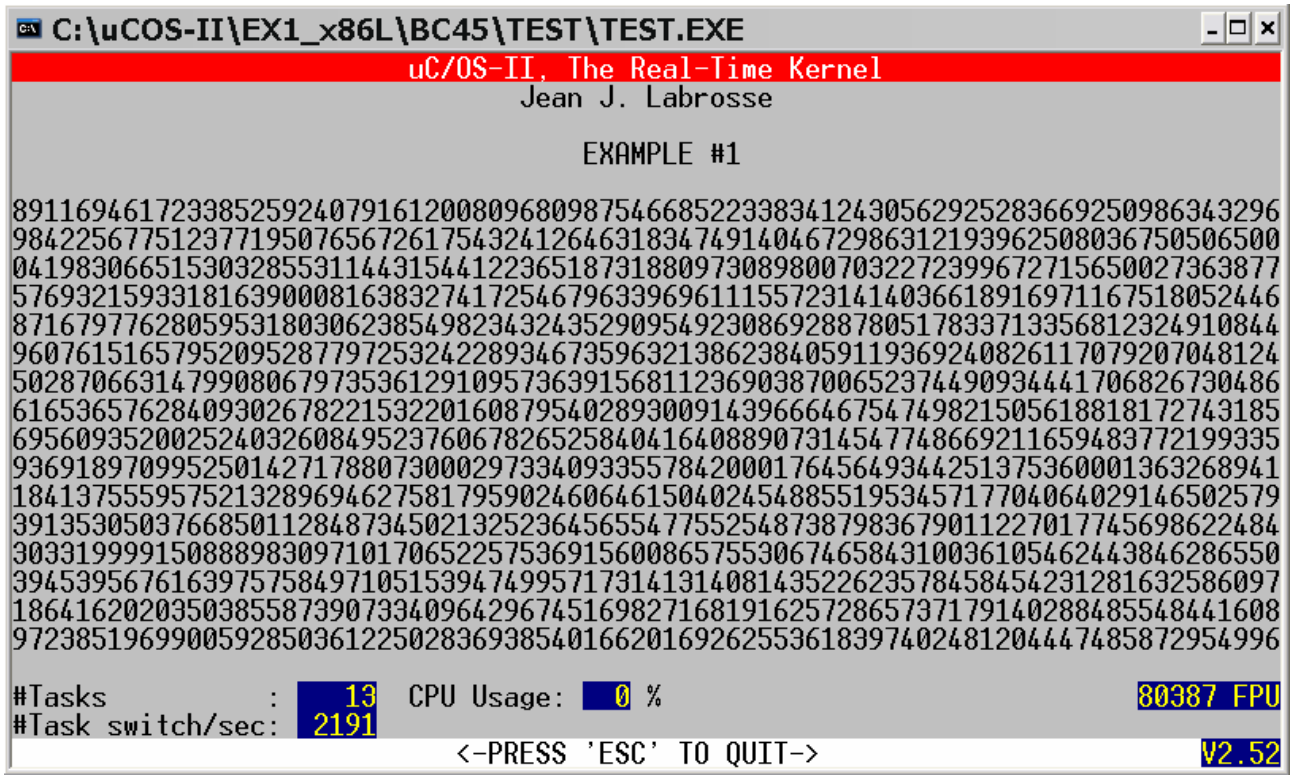
- Download the source code and emulator µC/OS-II
  - It is recommended to put the source code package "SOFTWARE" directly in C:\
- Test the first example
  - Execute C:\SOFTWARE\uCOS-II\EX1_x86L\BC45\TEST\TEST.EXE
  - Press ECS to leave
- Rename or remove the executable file
  - Rename TEST.EXE
- Compile 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 7
  - There is no space

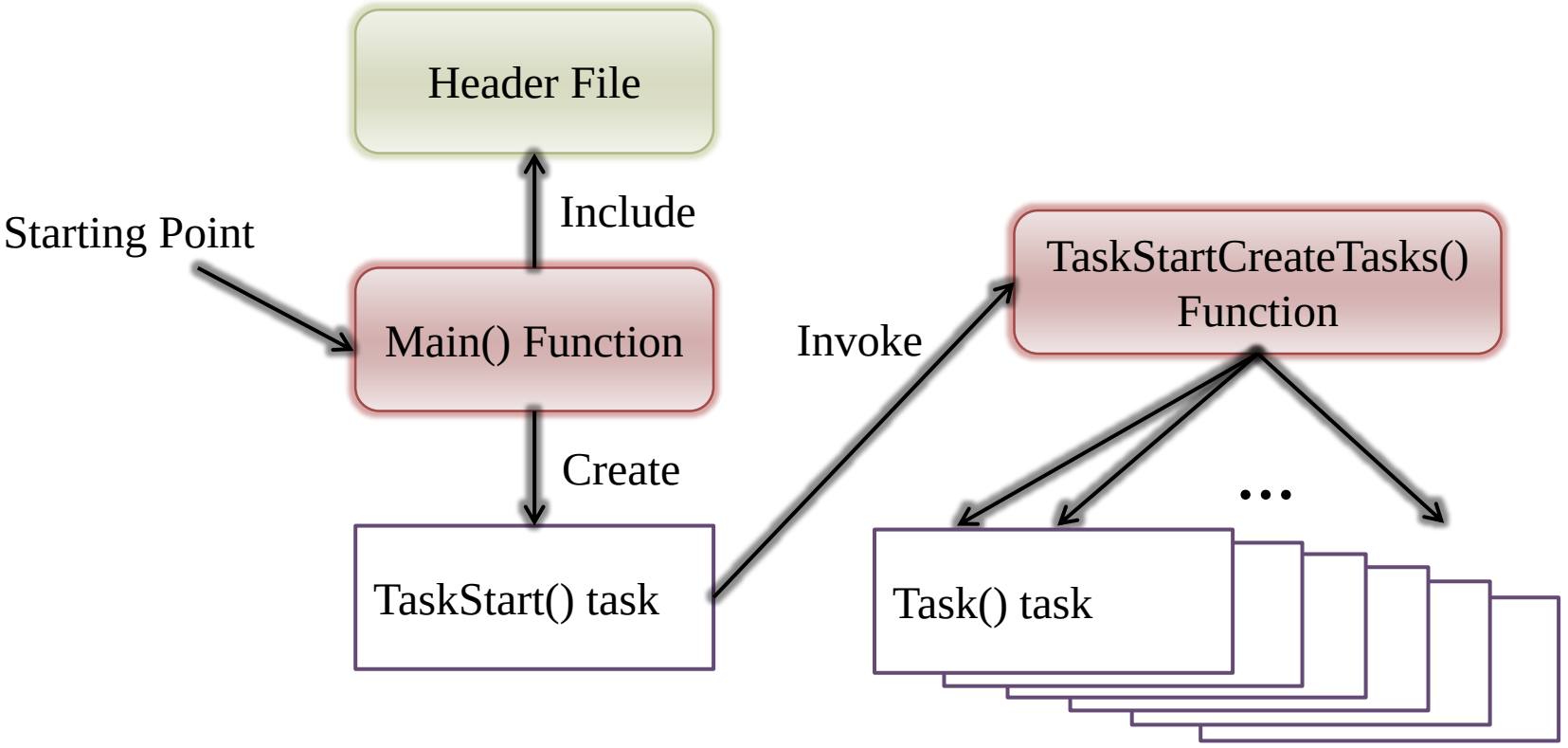
Example 1 on the Textbook

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

Real-Time Scheduling

CPU Scheduler

- Short-term scheduler selects a process among the processes in the ready queue, and allocates the CPU to the selected process
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is nonpreemptive
- All other scheduling is preemptive

Dispatcher

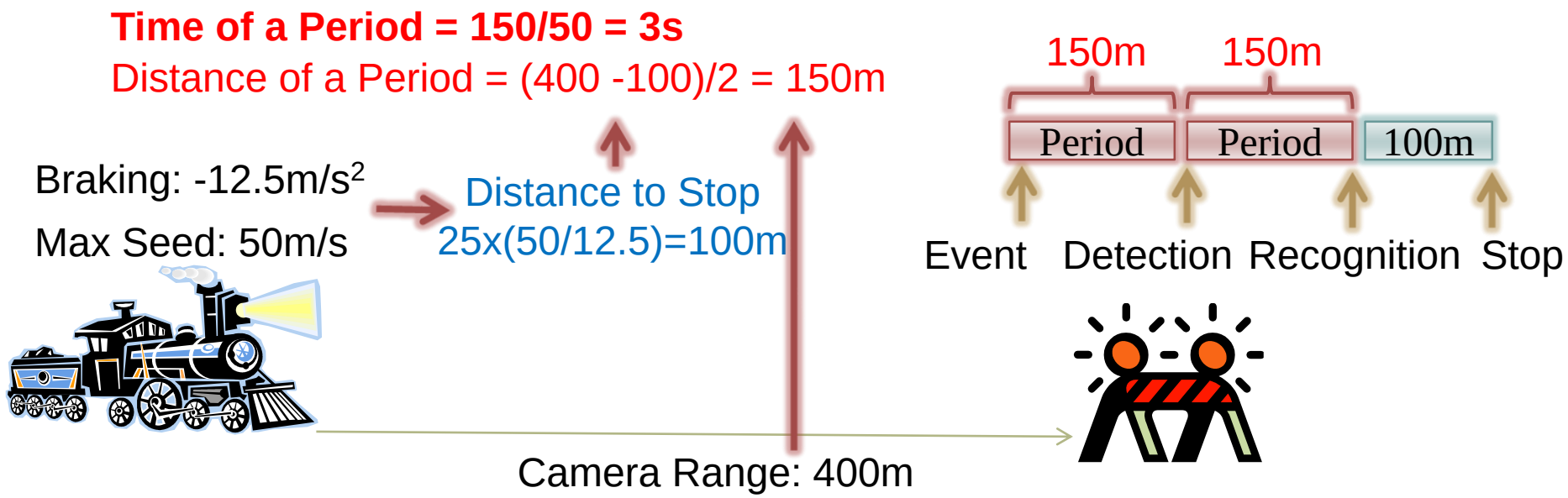
- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to resume that process
- Dispatch latency – the time it takes for the dispatcher to stop one process and start another running

Scheduling Algorithms

- First-Come, First-Served Scheduling (FIFO)
- Shortest-Job-First Scheduling (SJF)
- Priority Scheduling
- Round-Robin Scheduling (RR)
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling
- Multiple-Processor Scheduling

An Example of Real-Time Tasks

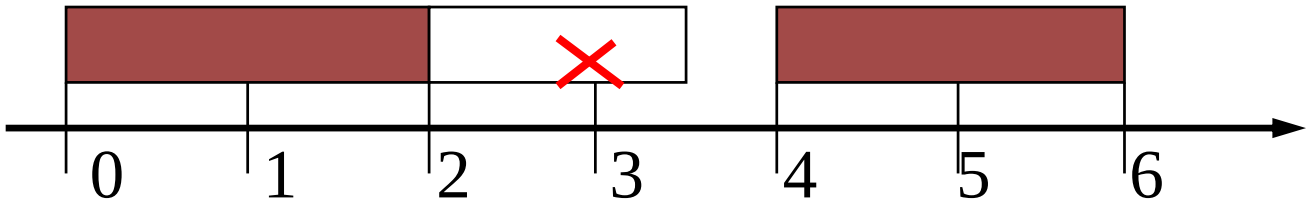
- A camera periodically takes a photo
- The image recognition result will be produced before the next period
- If there is an obstacle, the train automatically brakes



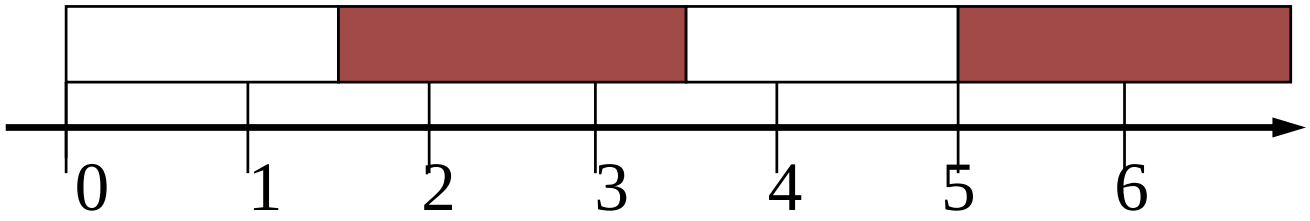
Periodic Task Scheduling

- Studying: 2 days per 4 days 
  Playing Basketball: 1.5 days per 3 days 

- Case 1: Studying is always more important



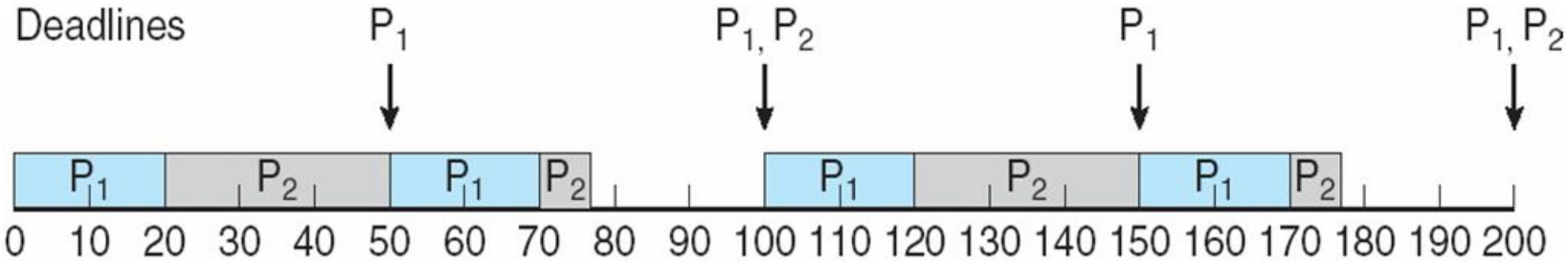
- Case 2: Doing whatever is more urgent



A Static Scheduling Algorithm—Rate Monotonic Scheduling

- A static priority is assigned to each task based on the inverse of its period
  - A task with shorter period ➔ higher priority
  - A task with longer period ➔ lower priority
  - For example:
    - P_1 has its period 50 and execution time 20
    - P_2 has its period 100 and execution time 37

    ➔P_1 is assigned a higher priority than P_2

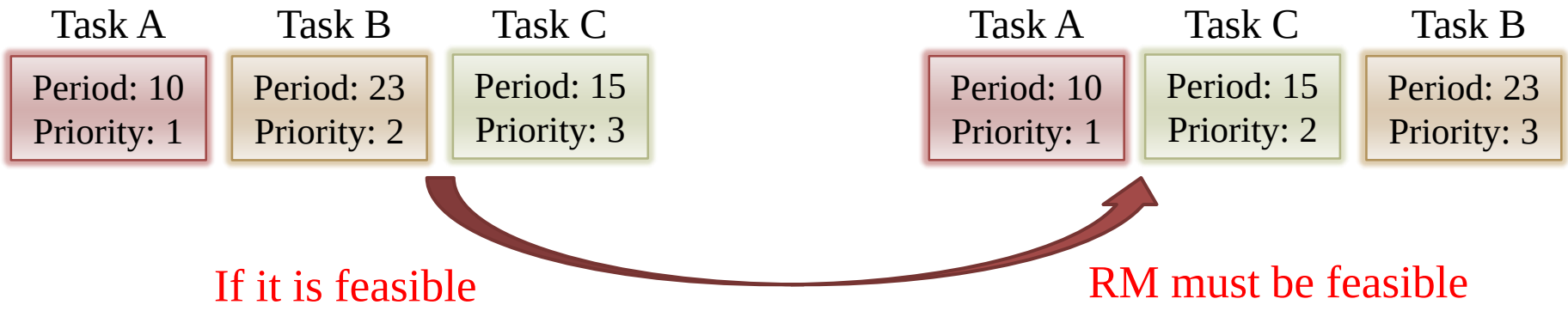


Property of Rate Monotonic Scheduling

- The **rate monotonic** (RM) priority assignment assigns processes priorities according to their request rates
  - If a feasible fixed priority assignment exists for some process set, then the rate monotonic priority assignment is feasible for that process set

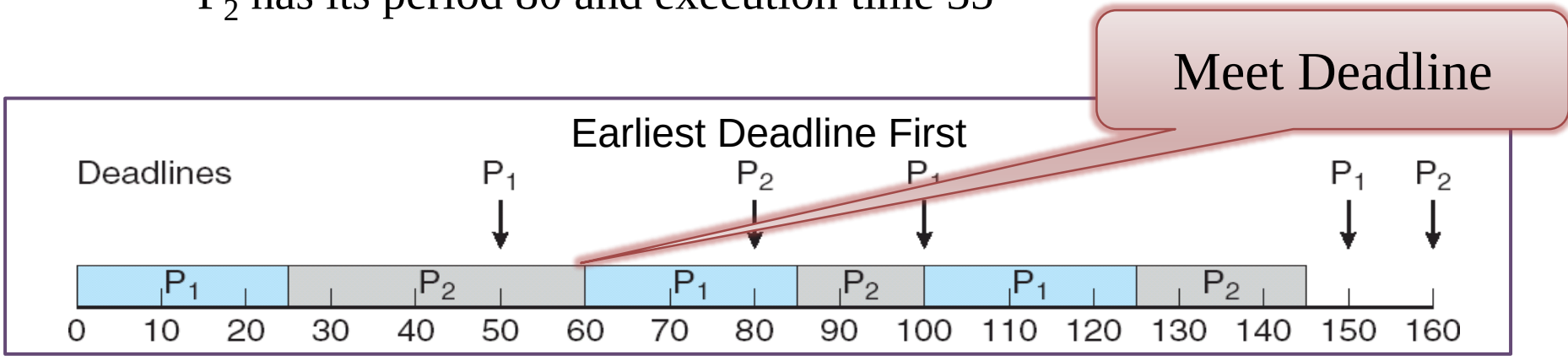
→ The optimal fixed priority assignment

Proof. Exchange the priorities of two tasks if their priorities are out of RMS order.



A Dynamic Scheduling Algorithm—Earliest Deadline First Scheduling

- Dynamic priorities are assigned according to deadlines
  - The earlier the deadline, the higher the priority
  - The later the deadline, the lower the priority
  - For example:
    - P_1 has its period 50 and execution time 25
    - P_2 has its period 80 and execution time 35



Project Requirements

Requirements

- Task Scheduling
  - Adopt priority-driven scheduling
  - The scheduler always schedules the highest priority ready task to run
  - Modify the priority of each task
  - Related code in uC/OS II
    - See OS_Sched( ) for scheduling policy
    - See OSTimeTick() for time management
    - See OSIntExit( ) for the interrupt management
- Provide the RM Scheduler
  - Input: A task set, each task is with its execution time and period
  - Output: The printed result of each task

Report and Source Files

- Report
  - File name: OSProjectStudentIDReport
  - File type: PDF
  - Only four A4 pages, 12 pt words
- Source File
  - File name: OSProjectStudentIDSource
  - File type: ZIP
  - Source code of your project (the whole SOFTWARE directory)
- Deadline is 20:00 2023/12/20
- Upload to the e-learning system

Grading

- Implementation
  - Install µC/OS-II and successfully compile a new application 30%
  - Run the RM scheduler with the given input files 30%
- Report
  - 20%
- Bonus
  - Implement EDF scheduler 10%
  - Use the semaphore functions of µC/OS-II and make a deadlock during the task running 10%
- Demo Q&A
  - 20%

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 total utilization is no more than 90%
- The number of tasks is no more than 7

Input Example (1/2)

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


Input Example (2/2)

```
5
1 18
1 17
2 16
1 20
1 6
```


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

Hints (1/2)

- You can read three other example in the document and refer to the source code.
- In order to implement a new scheduler, we might have to modify the os_tcb data structure to include some new attributes.
- The function OSTaskCreateExt() is used to create tasks, and we can modify this function to input the execution time and the period to each task.
- Each task executes an infinite loop and uses OSTimeGet() to get the execution time, where OS_TICKS_PER_SEC is the number of ticks for a second.
  - Note that a task might be preempted during its execution.
- Use OSTimeDly() when the task finish its execution.

Hints (2/2)

- Modify the deadline of a task before it call OSTimeDly() (ex: OSTCBCur->deadline= OSTCBCur->deadline+TaskPeriod)
- When the delay of a task is completed, the function OSTaskResume() is called to put the task back to ready queue and reschedule.
- Modify the function OS_Sched() to pick the task with the shortest period or the earliest deadline.
- OSStart() is used to start the execution of tasks.