Operating System Concepts

Che-Wei Chang

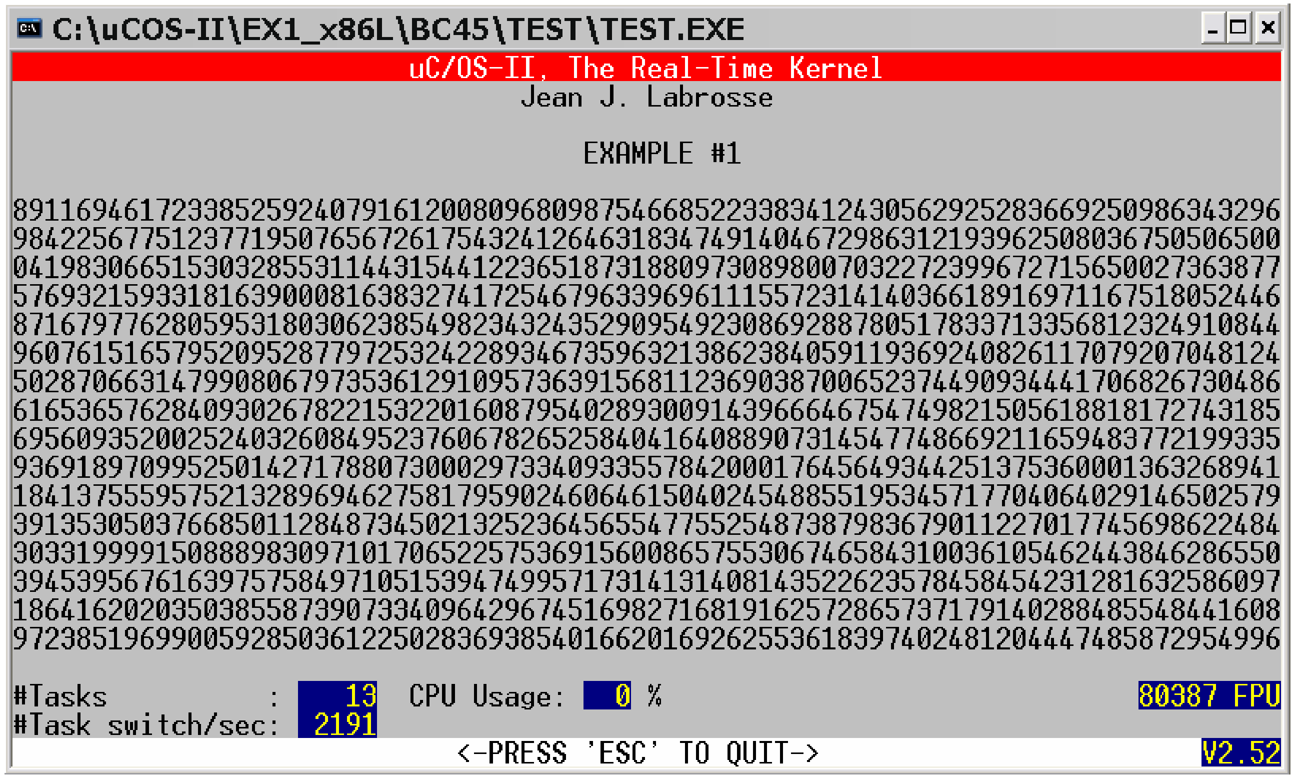
chewei@mail.cgu.edu.tw

Department of Computer Science and Information Engineering, Chang Gung University

Homework 5 (Final Project)-Scheduler Implementation on μC/OS-II

Example 1 on the Textbook

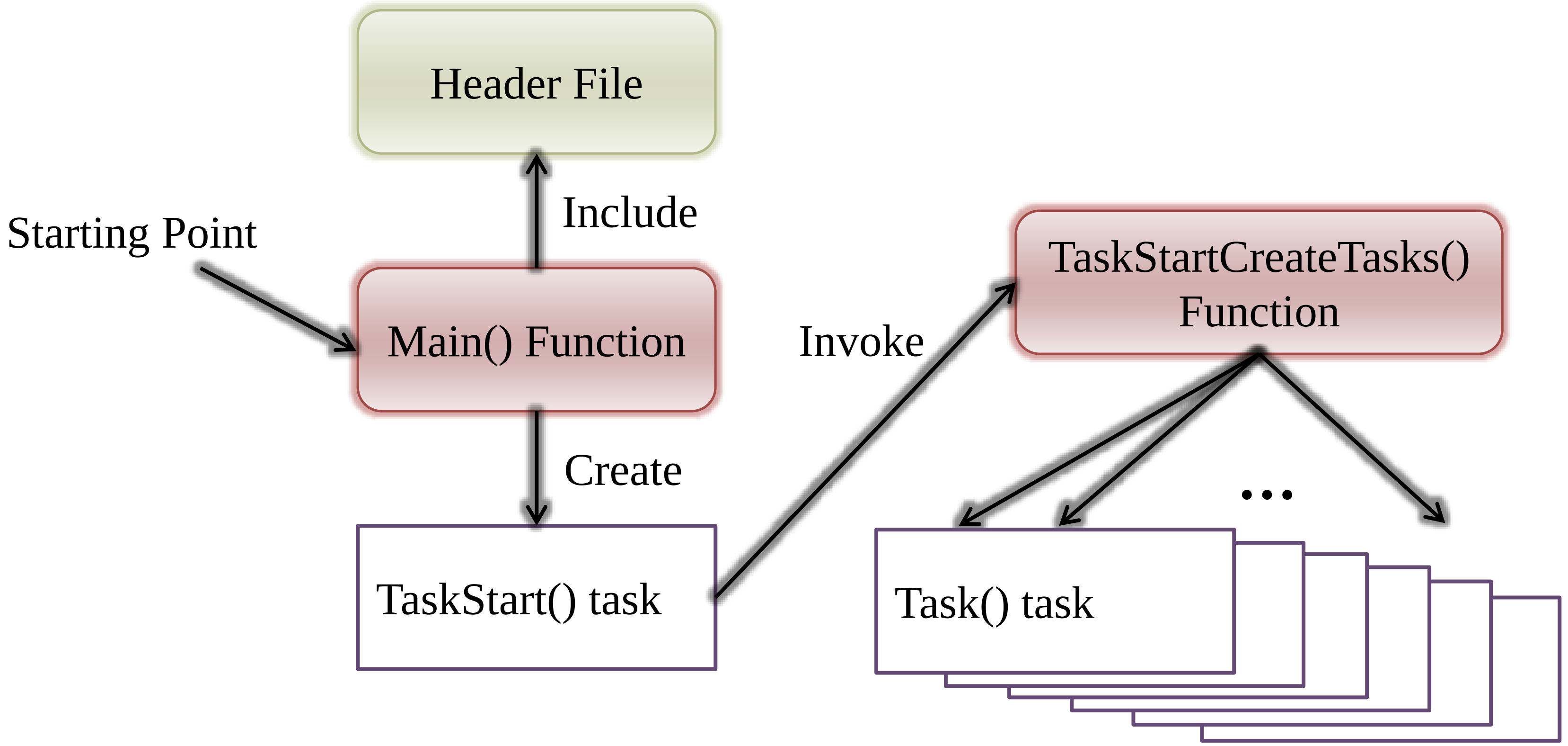
An Example on μC/OS-II: Multitasking



```
C:\uCOS-II\EX1_x86L\BC45\TEST\TEST.EXE
                         uC/OS-II, The Real-Time Kernel
                               Jean J. Labrosse

                                  EXAMPLE #1

89116946172338525924079161200809680987546685223383412430562925283669250986343296
98422567751237719507656726175432412646318347491404672986312193962508036750506500
04198306651530328553114431544122365187318809730898007032272399672715650027363877
57693215933181639000816383274172546796339696111557231414036618916971167518052446
87167977628059531803062385498234324352909549230869288780517833713356812324910844
96076151657952095287797253242289346735963213862384059119369240826117079207048124
50287066314799080679735361291095736391568112369038700652374490934441706826730486
61653657628409302678221532201608795402893009143966646754749821505618818172743185
69560935200252403260849523760678265258404164088907314547748669211659483772199335
93691897099525014271788073000297334093355784200017645649344251375360001363268941
18413755595752132896946275817959024606461504024548855195345717704064029146502579
39135305037668501128487345021325236456554775525487387983679011227017745698622484
30331999915088898309710170652257536915600865755306746584310036105462443846286550
39453956761639757584971051539474995717314131408143522623578458454231281632586097
18641620203503855873907334096429674516982716819162572865737179140288485548441608
97238519699005928503612250283693854016620169262553618397402481204447485872954996

#Tasks          :      13  CPU Usage:   0 %                              80387 FPU
#Task switch/sec:    2191
                            <-PRESS 'ESC' TO QUIT->                           V2.52
```

- 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

Project Requirements

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

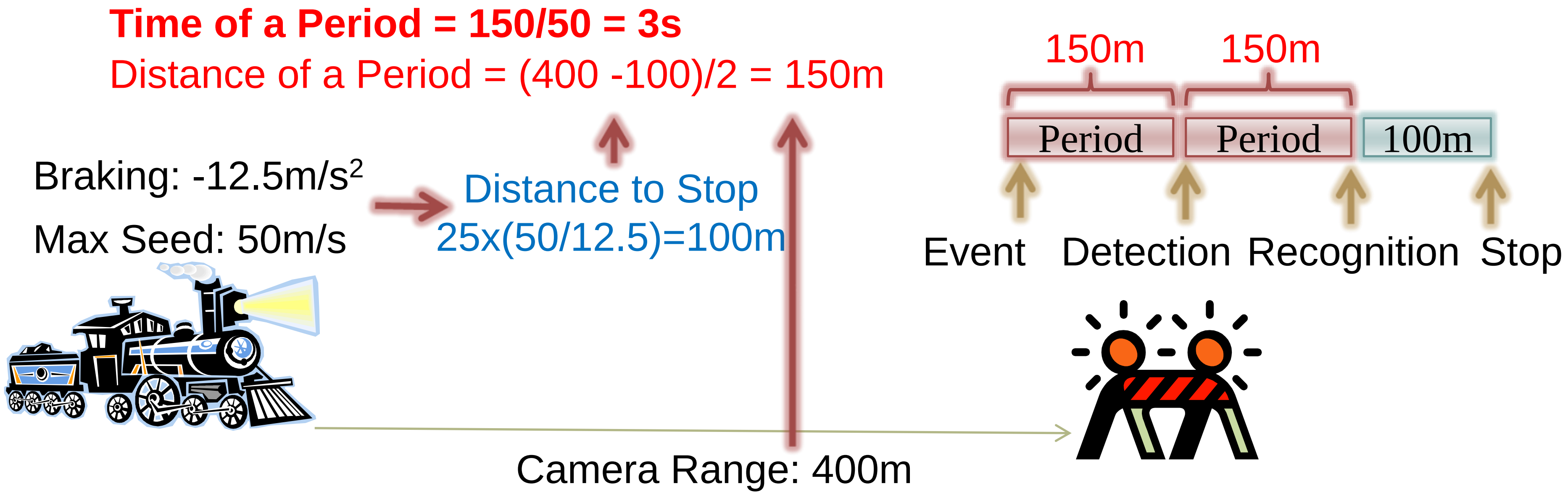
- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to resume that process
- Dispatch latency – the time it takes for the dispatcher to stop one process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

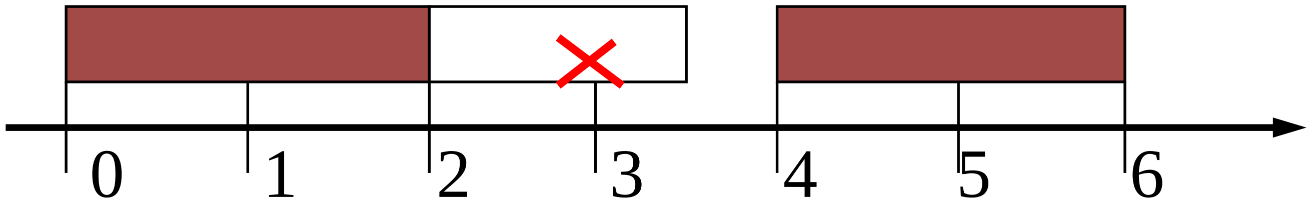
- A camera periodically takes a photo
- The image recognition result will be produced before the next period
- If there is an obstacle, the train automatically brakes



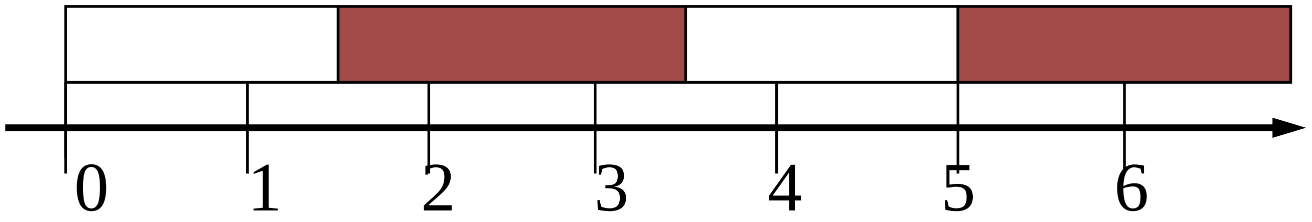
Periodic Task Scheduling

- 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



Project Requirements

- Read the input file and create the periodic tasks
- Implement the SJF scheduling (執行時間短的優先，如果執行時間一樣，周期短的優先。如果都一樣，誰先都可以)
- Bonus 1 (10%): Implement the preemptive SFJ scheduling
- Bonus 2 (10%): Implement the RR scheduling (time quantum = 2)
- Bonus 3 (10%): Count and show the waiting time of each task instance

Input File Format

- The total required CPU time of all tasks are no more than 80%
- File format: (all are integers)

(number of tasks)

(period of task 1) (execution time of task 1)

(period of task 2) (execution time of task 2)

(period of task 3) (execution time of task 3)

⋮

Sample Input File

```
4
12 2
10 1
5 1
16 3
```

Report

1. The steps for your implementation
2. The problem you met, and how you solved it
3. The bonus you have done
4. <u>**The reference of this homework**</u>

- The report is limited within ***6*** pages in PDF
- Each bonus you have done, one more page for the report

Grading

- Implementation
  - Periodic tasks 30%
  - SJF scheduling 30%
- Report
  - 20%
- Bonus
  - Bonus 1 10%
  - Bonus 2 10%
  - Bonus 3 10%
- Demo Q&A
  - 20%

Submission

- Homework 5 deadline: at 20:00 on 2023-1-10

  ➔**NO DELAY!**
- Upload to e-learning system
- The title of the report: OSHomework5StudentID
- **Point deduction for wrong format: 10%**

➔DEMO will be arranged!

實作流程參考 1/2

- 先想一遍，然後翻看電子書相關的功能敘述。
- 建議早點開始做，老師會不定期給實作提示，有開始做的人才會聽得懂提示。
- 用多層for loop實作出週期性工作該有的執行時間。
  - 因為每個人的電腦能力不同，所以要執行多少工作才會消耗一秒鐘，因電腦而異。所以得要手動測出來。
  - Demo的時候請用跟寫作業時同一台電腦，免得時間有大量偏差。
- 參考第五章的知識算出當工作執行完後，應該要睡多久才是到下個週期的開始時間。
  - 到此為止先測試一下一個以及多個周期性工作是否可以正常運作，可以透過在螢幕上顯示些資訊，
- 再次確認自己真的知道什麼是SJF

實作流程參考 2/2

- 找出μC/OS-II排程器所在的位置並做修改。
  - 預設是挑選ready且優先權高的工作(void OS_Sched (void))，改成挑選ready且執行時間短的工作。
  - 實作方式有兩種：
    - 讀檔的時候就依照工作的實行時間長短去指派工作的優先權，如此一來就不需要修改排程器(較容易實作)，要確void OS_Sched (void)的時間點對不對，如果此時不需要重排工作，就要跳過呼叫。
    - 修改排程器的程式碼，改為選實行時間短的工作優先(後續加分題比較好做)。
- Preemptive SFJ: 找interrupt處理的地方，當有工作ready後，查看是否執行時間比現正執行的工作還要短 。
- RR: 依序執行每個ready的工作。當一個工作開始執行時，作業系統為其倒數一個time quantum的時間。如果換工作了就重新倒數，如果倒數到歸零則強制換到下一個。