Embedded OS Implementation, Fall 2020

Project #1 (due November 11, 2020 (Wednesday) 13:00)

[PART I] Task Control Block Linked List

Objective:

Follow the previous homework (HW1), please add some code to the μC/OS-II scheduler **in the kernel level** to observe the operations of task control block (TCB) and TCB linked list. linklist

※The TCB address is dynamic

The output results are shown below:

```
OSTick    created, Thread ID 21332
Task[ 63] created, TCB Address 0010F660
-------After TCB[63] being linked-------
Previous TCB point to address 00000000
Current  TCB point to address 0010F660
Next     TCB point to address 00000000

Task[  1] created, TCB Address 0010F6B8
-------After TCB[1] being linked-------
Previous TCB point to address 00000000
Current  TCB point to address 0010F6B8
Next     TCB point to address 0010F660

Task[  2] created, TCB Address 0010F710
-------After TCB[2] being linked-------
Previous TCB point to address 00000000
Current  TCB point to address 0010F710
Next     TCB point to address 0010F6B8

================= TCB linked list =================
Task     Prev_TCB_addr    TCB_addr         Next_TCB_addr
2        00000000         0010F710          0010F6B8
1        0010F710         0010F6B8          0010F660
63       0010F6B8         0010F660          00000000
```

[PART II] RM Scheduler Implementation

Objective:

To implement the Rate Monotonic (RM) scheduler for periodic tasks, and observe the scheduling behaviors.

Problem Definition:

Implement the following four task sets of periodic tasks. Add necessary code to the μC/OS-II scheduler **in the kernel level** to observe how the task suffers the schedule delay.

Periodic Task Set = {τ_{ID} (arrival time, execution time, period)}

Task set 1 = {τ_1 (0, 1, 3), τ_2 (0, 3, 6)}

Task set 2 = {τ_1 (0, 8, 15), τ_2 (0, 2, 5)}

Task set 3 = {τ_1 (1, 1, 3), τ_2 (0, 4, 6)}

Task set 4 = {τ_1 (0, 4, 6), τ_2 (2, 2, 10), τ_3 (1, 1, 5)}

※ The priority of task is set according to the RM scheduling rules.

Evaluation:

The output format:

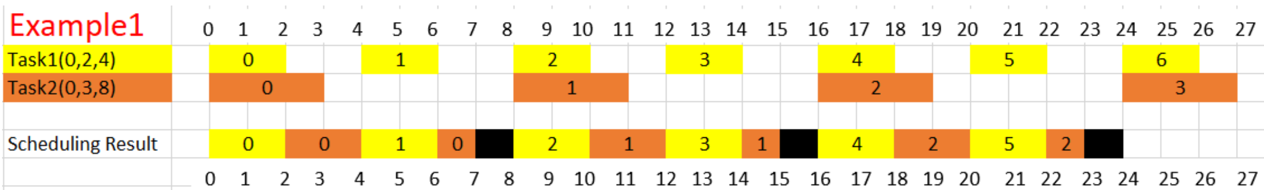
Tick	Event	CurrentTask ID	NextTask ID	ResponseTime	# of ContextSwitch
##	Preemption	task(ID)(job number)	task(ID)(job number)		
##	Completion	task(ID)(job number)	task(ID)(job number)	##	##
##	MissDeadline	task(ID)(job number)	-------------------		

※ If task is Idle Task, print "*task(priority)*".

The Example1 of output results:

Consider two tasks τ_1 (0,2,4) and τ_2 (0,3,8) with priority 1 and 2, respectively.

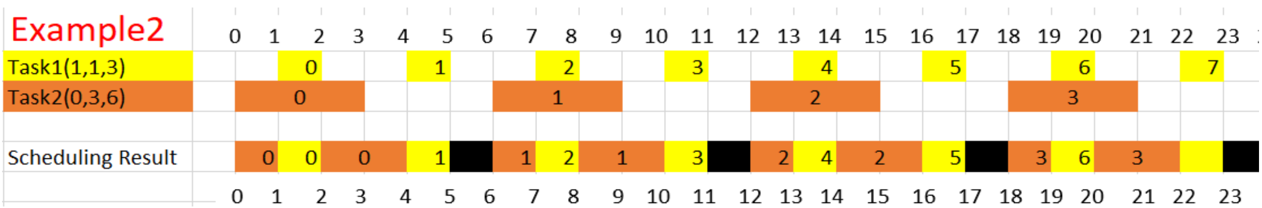
Tick	Event	CurrentTask ID	NextTask ID	ResponseTime	# of ContextSwitch
2	Completion	task(1)(0)	task(2)(0)	2	1
4	Preemption	task(2)(0)	task(1)(1)		
6	Completion	task(1)(1)	task(2)(0)	2	2
7	Completion	task(2)(0)	task(63)	7	4
8	Preemption	task(63)	task(1)(2)		
10	Completion	task(1)(2)	task(2)(1)	2	2
12	Preemption	task(2)(1)	task(1)(3)		
14	Completion	task(1)(3)	task(2)(1)	2	2
15	Completion	task(2)(1)	task(63)	7	4
16	Preemption	task(63)	task(1)(4)		
18	Completion	task(1)(4)	task(2)(2)	2	2
20	Preemption	task(2)(2)	task(1)(5)		
22	Completion	task(1)(5)	task(2)(2)	2	2
23	Completion	task(2)(2)	task(63)	7	4



The Example2 of output results:

Consider two tasks τ_1 (1,1,3) and τ_2 (0,3,6) with priority 1 and 2, respectively.

Tick	Event	CurrentTask ID	NextTask ID	ResponseTime	# of ContextSwitch
1	Preemption	task(2)(0)	task(1)(0)		
2	Completion	task(1)(0)	task(2)(0)	1	2
4	Completion	task(2)(0)	task(1)(1)	4	3
5	Completion	task(1)(1)	task(63)	1	2
6	Preemption	task(63)	task(2)(1)		
7	Preemption	task(2)(1)	task(1)(2)		
8	Completion	task(1)(2)	task(2)(1)	1	2
10	Completion	task(2)(1)	task(1)(3)	4	4
11	Completion	task(1)(3)	task(63)	1	2
12	Preemption	task(63)	task(2)(2)		
13	Preemption	task(2)(2)	task(1)(4)		
14	Completion	task(1)(4)	task(2)(2)	1	2
16	Completion	task(2)(2)	task(1)(5)	4	4
17	Completion	task(1)(5)	task(63)	1	2
18	Preemption	task(63)	task(2)(3)		
19	Preemption	task(2)(3)	task(1)(6)		
20	Completion	task(1)(6)	task(2)(3)	1	2



Crediting:

[PART I] Task Control Block Linked List [20%]

- The screenshot results. (10%)
- A report that describes your implementation (please attach the screenshot of the code and **MARK** the modified part). (10%)

[PART II] RM Scheduler Implementation [80%]

- The screenshot results (with the given format) of four task sets. (Time ticks 0-30 or miss deadline). (40%)
- A report that describes your implementation (please attach the screenshot of the code and **MARK** the modified part). (40%)

※You must modify source code!

Project submit:

Submit to Moodle.

Submit deadline: November 11, 2020 (Wednesday) 13:00

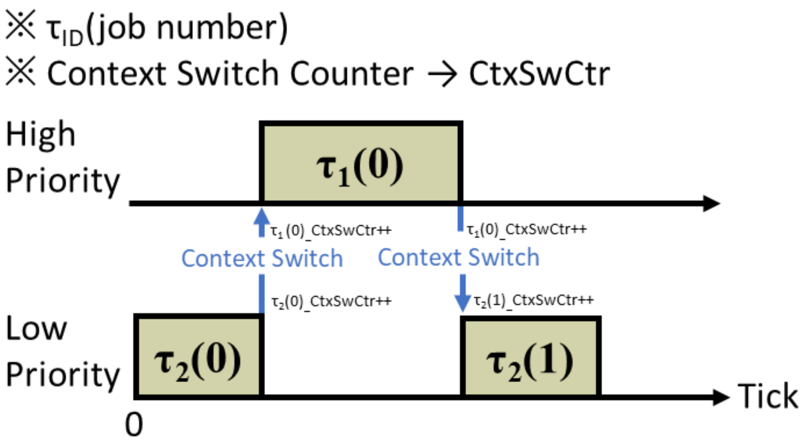
File name format: RTOS_your student ID_PA1.zip

RTOS_ your student ID_PA1.zip includes:

- The report (RTOS_ your student ID_PA1.pdf).
- The file you modify(main.c, os_core.c, etc.)

Hints:

1. Task (**a, c, p**) means that a task arrives at **a** tick and then executes **c** tick in every **p** ticks.
2. We define the number of context switch for every task in every period is the sum of switch in and switch out.



3. Please modify **OS_TASK_STAT_EN** and **OS_TMR_EN** from **1** to **0** in "**os_cfg.h**" file to disable the statistic task and timer management.

```
#define OS_TASK_QUERY_EN          1u   /*
#define OS_TASK_REG_TBL_SIZE      1u   /*
#define OS_TASK_STAT_EN           0u   /*
#define OS_TASK_STAT_STK_CHK_EN   1u   /*
```

```
#define OS_TMR_EN                 0u   /*
#define OS_TMR_CFG_MAX           16u   /*
#define OS_TMR_CFG_NAME_EN        1u   /*
#define OS_TMR_CFG_WHEEL_SIZE     7u   /*
```

4. We build a simple task set at the application region (i.e., main.c), and it includes some simple periodic tasks (c, p). A simple periodic task as following：

while(1){

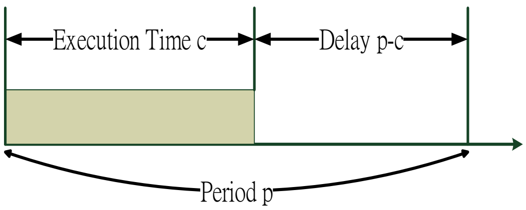
Do something →A

Wait for next period →B

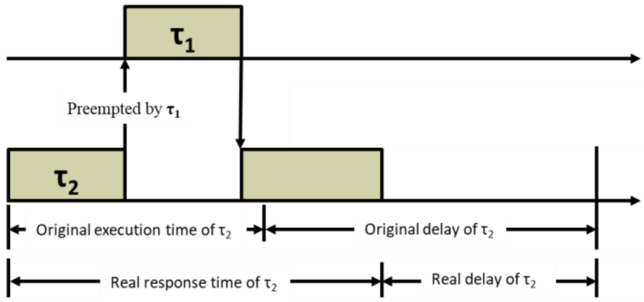
}

The periodic task can be subdivided into two parts. The part A is in charge of executing some operations, the execution time c is estimated in offline profiling. The part B is in charge of waiting for the next arrival time, the waiting time is p-c. Therefore, if there is an application we can model it as a simple task to simulate its behavior.

```
while(1){
      Start = OSTimeGet() ;
      While(OStimeGet() - start < c){              →A
            //Do something
      }
      OSTimeDly (p-c) ;                            →B
}
```



However, **if two more tasks exist in the system, this task function might not be adaptive**. That is because task preemption occurs at runtime to affect the set time in task function. For example, the following figure shows the task preemption occurring between the task τ_1 and τ_2 execution.



The original execution time and delay of τ_2 you set will mismatch, because the original execution time is affected by τ_1 (i.e., the execution time of τ_2 will include the execution time of τ_1.).

In min function which is in application region (i.e., main.c). There is a function named **OSTaskCreateExt(...)** which is the function creates a task. There are lots of data structures used in this function. If you have no clue where to start this project, you can trace this function.

```
main(){
      ...
      OSTaskCreateExt(Task1, ...) ;
      OSTaskCreateExt(Task2, ...) ;
      ...
      OSStart();
}
```

5. To add new variables into the task control block for recoding the execution time, job ID, etc. You can trace the function **OS_TCBInit(...)**.