Embedded OS Implementation, Fall 2022

Project #3 (due Dec. 18, 2022 (Sunday) 12:00)

[PART I] NPCS Implementation

Objective:

Implement the non-preemptible critical section (NPCS) based on the RM scheduler in uC/OS-II.

Problem Definition:

uC/OS-II uses a variation of the priority inheritance protocol to deal with priority inversions. In this assignment, you are going to implement the NPCS based on the RM scheduler in uC/OS-II.

Consider the two examples and observe how the task suffers the scheduler delay.

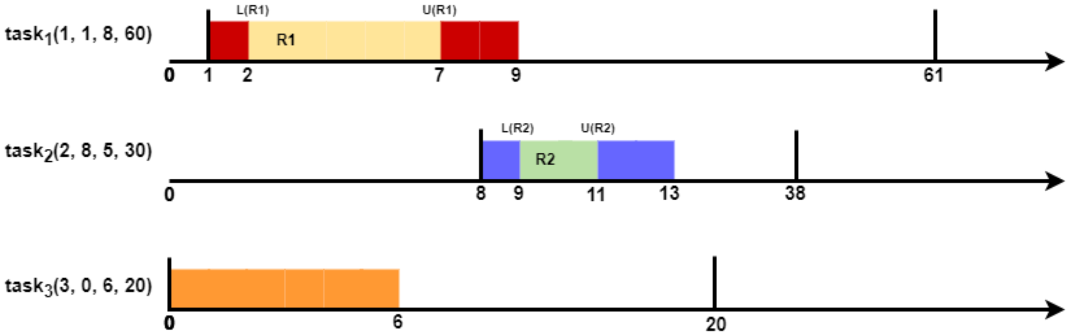
Periodic Task Set = { task$_{ID}$ (ID, arrival time, execution time, period, R1 lock, R1 unlock, R2 lock, R2 unlock) }

※ L(R#): Lock resource #, U(R#): Unlock resource #

Example Task Set 1 = { task$_1$ (1, 1, 8, 60, 1, 6, 0, 0),

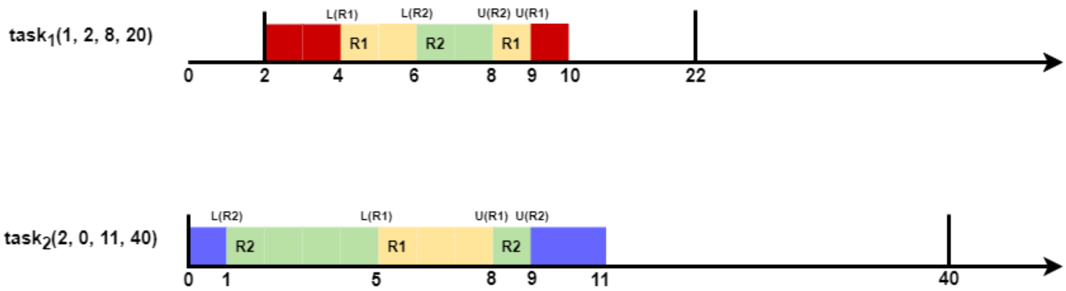
task$_2$ (2, 8, 5, 30, 0, 0, 1, 3),

task$_3$ (3, 0, 6, 20, 0, 0, 0, 0) }



Example Task Set 2 = { task$_1$ (1, 2, 8, 20, 2, 7, 4, 6),

task$_2$ (2, 0, 11, 40, 5, 8, 1, 9)}



The input file format:

Task ID	Arrival Time	Execution Time	Task Period	R1 Lock Time	R1 Unlock Time	R2 Lock Time	R2 Unlock Time
##	##	##	##	##	##	##	##

Example of the input file:


TaskSet1 - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V)

```
1 1 8 60 1 6 0 0
2 8 5 30 0 0 1 3
3 0 6 20 0 0 0 0
```

※ Lock time and unlock time are relative to the task start time.

Evaluation:

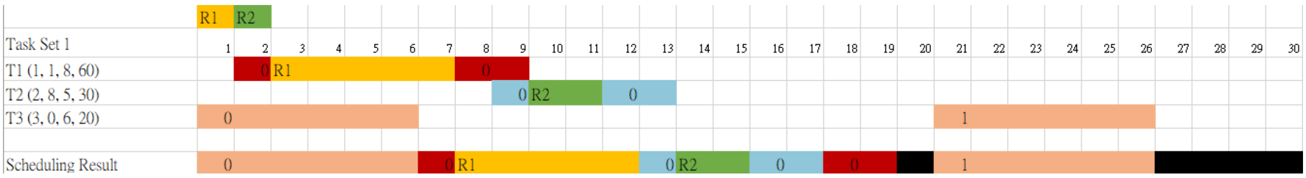
The output format:

Tick	Event	CurrentTask ID	NextTask ID
##	Preemption	task(ID)(job number)	task(ID)(job number)
##	Completion	task(ID)(job number)	task(ID)(job number)

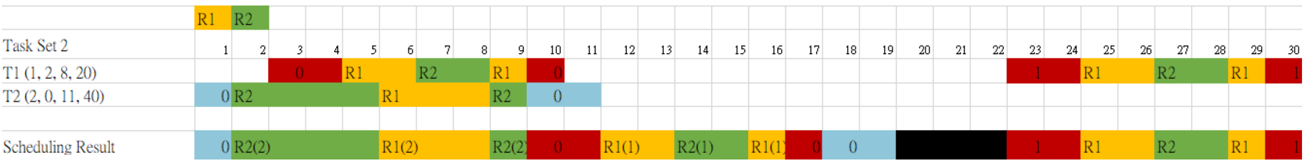
※ When getting/releasing shared resources, please follow the format:

Tick	Task ID	Event	Resource
##	task#	get/release	R#

The output results of Example 1:



The output results of Example 2:



[PART II] CPP Implementation

Objective:

Implement the ceiling-priority protocol (CPP) based on the RM scheduler in uC/OS-II.

Problem Definition:

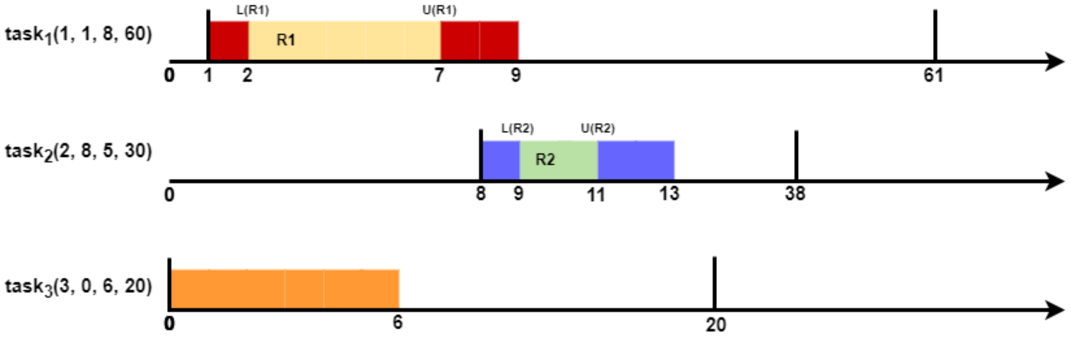
uC/OS-II uses a variation of the priority inheritance protocol to deal with priority inversions. In this assignment, you are going to implement the CPP based on the RM scheduler in uC/OS-II.

Consider the two examples and observe how the task suffers the scheduler delay.

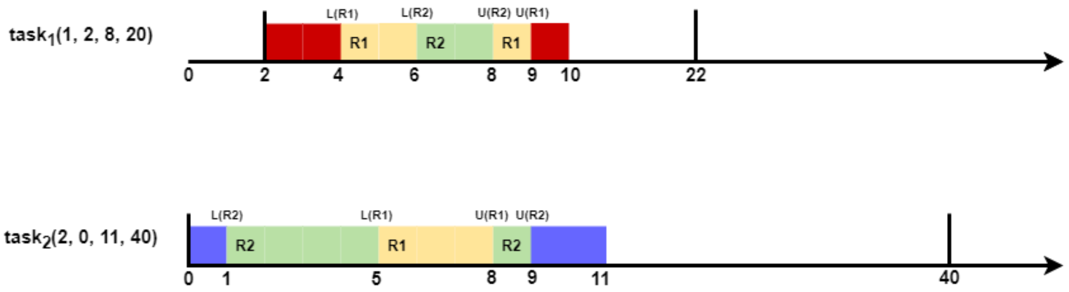
Periodic Task Set = { task$_{ID}$ (ID, arrival time, execution time, period, R1 lock, R1 unlock, R2 lock, R2 unlock) }

※ L(R#): Lock resource #, U(R#): Unlock resource #

Example Task Set 1 = { task$_1$ (1, 1, 8, 60, 1, 6, 0, 0),
task$_2$ (2, 8, 5, 30, 0, 0, 1, 3),
task$_3$ (3, 0, 6, 20, 0, 0, 0, 0) }



Example Task Set 2 = { task$_1$ (1, 2, 8, 20, 2, 7, 4, 6),
task$_2$ (2, 0, 11, 40, 5, 8, 1, 9)}



Evaluation:

The output format:

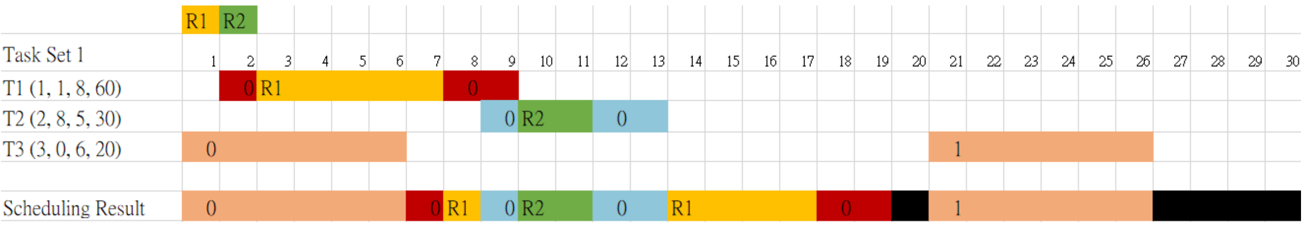
Tick	Event	CurrentTask ID	NextTask ID
##	Preemption	task(ID)(job number)	task(ID)(job number)
##	Completion	task(ID)(job number)	task(ID)(job number)

※ When getting/releasing shared resources, please follow the format:

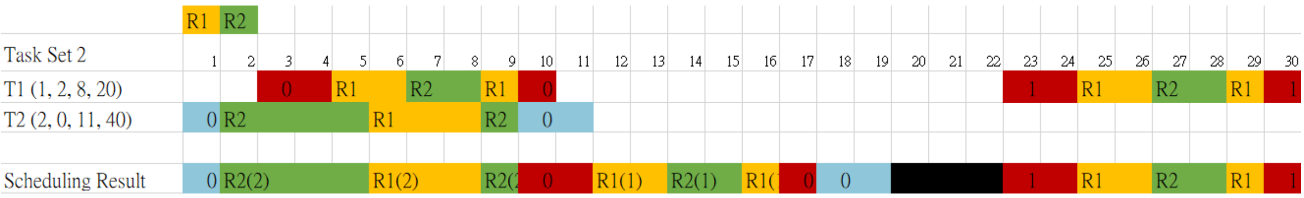
Tick	Task ID	Event	Resource	Priority Inheritance
##	task#	get/release	R#	## to ##

※ An odd number sets the resource's ceiling, and an even number forms the task's priority.

The output results of Example 1:



The output results of Example 2:



Credit:

[PART I] NPCS Implementation [40%]

- The correctness of schedule results of examples. Note the testing task set might not be the same as the given example task set. (20%)
- A report that describes your implementation (please attach the screenshot of the code and **MARK** the modified part). (20%)

[PART II] CPP Implementation [50%]

- The correctness of schedule results of examples. Note the testing task set might not be the same as the given example task set. (20%)
- A report that describes your implementation (please attach the screenshot of the code and **MARK** the modified part). (30%)

[PART III] Performance Analysis [10%]

- Compare the scheduling behaviors between NPCS and CPP with PART I and PART II results. (5%)
- Explain how NPCS and CPP avoid the deadlock problem. (5%)

※ **You must modify the source code.**

※ **Standard input and output filenames in the project are necessary for the checker. Please check the file names before submitting.**

```
#define INPUT_FILE_NAME "./TaskSet.txt"
#define OUTPUT_FILE_NAME "./Output.txt"
```

※ **Please set the parameter, INFO, as 10 to read more task information.**

```
#define INFO 10
```

※ **Please set the system end time as 100 seconds in this project.**

```
#define SYSTEM_END_TIME 100
```

※ **You must check your project can produce the correct output file.**

※ **We only use two share resources in this project.**

※ **We will use different task sets to verify your code.**

※ **You will submit two μC/OS-II projects for PART I and PART II, respectively.**

Project submit:

Submit to Moodle

Submit deadline: Dec. 18, 2022 (Sunday) 12:00

File name format: RTOS_Myyyddxxx_PA3.zip

RTOS_Myyyddxxx_PA3.zip includes:

- The report (RTOS_Myyyddxxx_PA3.pdf).
- Folder with the executable μC/OS-II project (**RTOS_Myyyddxxx_PA3_NPCS**).
- Folder with the executable μC/OS-II project (**RTOS_Myyyddxxx_PA3_CPP**).

※ Plagiarizing is strictly prohibited.

Hints:

1. In the application region, we define the priorities of tasks and shared resources.

```
#define R1_PRIO 1
#define R2_PRIO 3
#define TASK1_PRIORITY      2
#define TASK2_PRIORITY      4
```

2. We also declare shared resources, as follows:

```
OS_EVENT* R1;
OS_EVENT* R2;
```

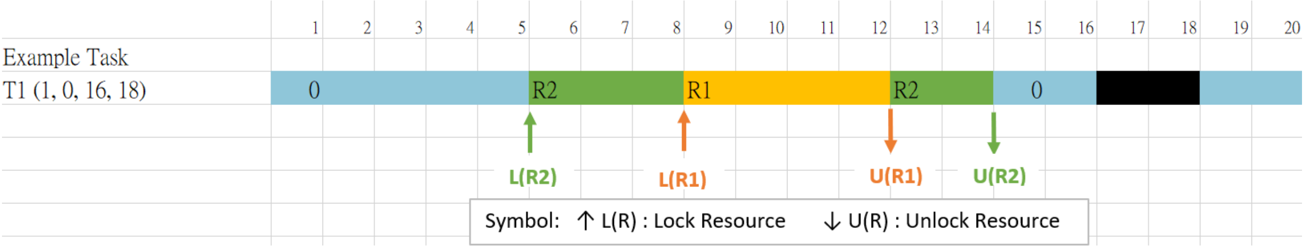
3. In the main function, we not only create tasks but also create shared resources.

```
INT8U err;
R1 = OSMutexCreate(R1_PRIO, &err);
R2 = OSMutexCreate(R2_PRIO, &err);
```

4. To simulate the duration that a resource is held, we can program a function to implement it:

```
void mywait(int tick)
{
#if OS_CRITICAL_METHOD==3
    OS_CPU_SR cpu_sr = 0;
#endif
    int now, exit;
    OS_ENTER_CRITICAL();
    now = OSTimeGet();
    exit = now + tick;
    OS_EXIT_CRITICAL();
    while (1) {
        if (exit <= OSTimeGet())
            break;
    }
}
```

5. To model a task's behavior, we can program the task function as following:




```c
void task1(void* pdata)
{
    INT8U err;
    while (1)
    {
        printf("%d\tTask 1\n", OSTimeGet());
        mywait(3);
        printf("%d\tTask 1 get R2\n", OSTimeGet());
        OSMutexPend(R2, 0, &err);
        mywait(7);

        printf("%d\tTask 1 get R1\n", OSTimeGet());
        OSMutexPend(R1, 0, &err);
        mywait(2);

        printf("%d\tTask 1 release R1\n", OSTimeGet());
        OSMutexPost(R1);
        mywait(1);

        printf("%d\tTask 1 release R2\n", OSTimeGet());
        OSMutexPost(R2);
        mywait(3);
        OSTimeDly( T1_Deadline - OSTimeGet());
    }
}
```