

Embedded OS Implementation

Project #3 (due Dec. 11, 2025 (Thursday) 8:00)

[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) }

Example Task Set 1 = { task₁ (1, 2, 6, 15, 1, 4, 2, 5),
task₂ (2, 0, 7, 20, 5, 6, 1, 3) }

Example Task Set 2 = { task₁ (1, 1, 8, 32, 1, 6, 0, 0),
task₂ (2, 8, 5, 30, 0, 0, 0, 0),
task₃ (3, 0, 4, 20, 0, 0, 1, 3) }

The input file format:

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

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

Evaluation:

The output format:

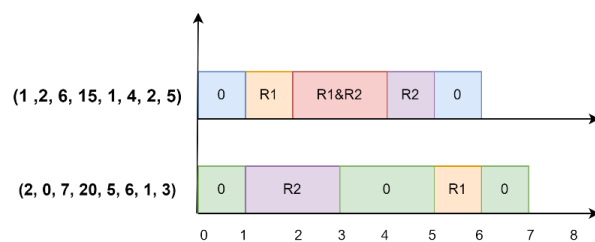
[illegible]

Blocking Time: Time blocked by task which priority lower than itself.

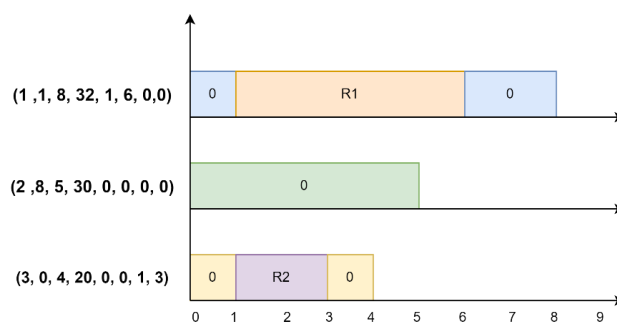
Preemption Time: Time blocked by task which priority **higher** than itself.

Event order: Unlock > Preemption = Completion > Lock > running

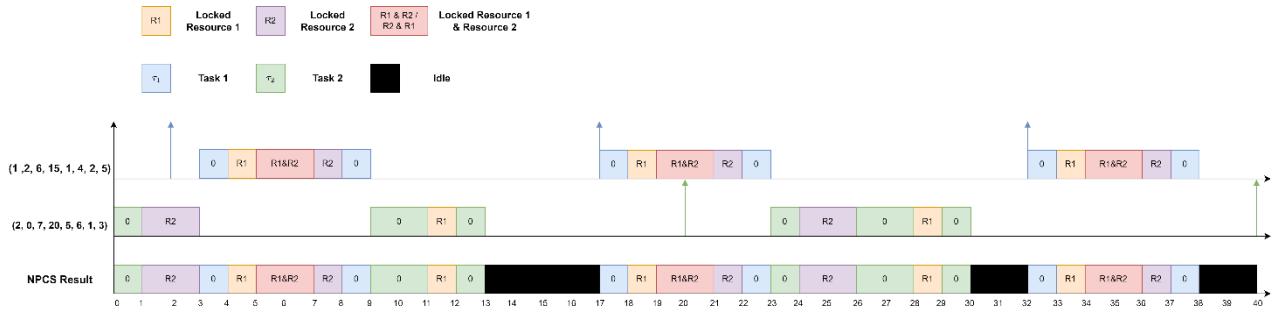
The Taskset of **Example 1**:



The Taskset of **Example 2**:



The output results of **Example**

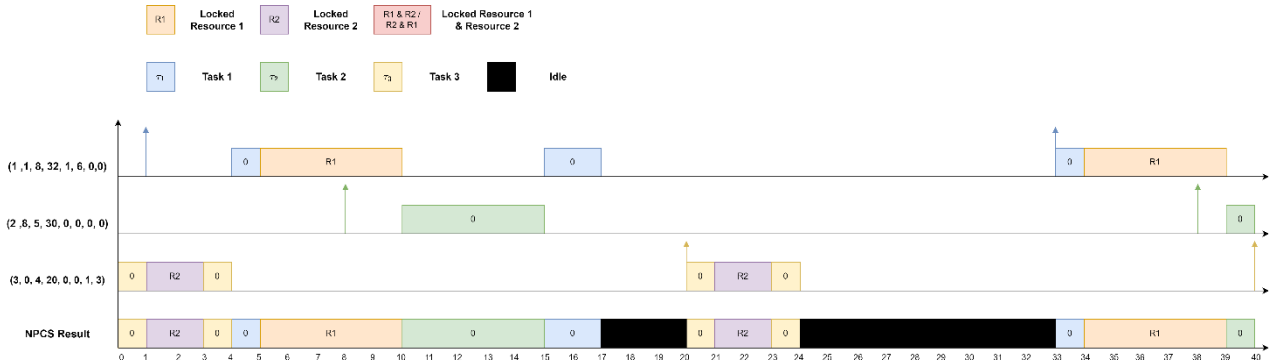


```

0 task( 2) is running
1 LockResource task( 2)( 0) R2
1 task( 2) is running
2 task( 2) is running
3 UnlockResource task( 2)( 0) R2
3 Preemption task( 2)( 0) task( 1)( 0)
3 task( 1) is running
4 LockResource task( 1)( 0) R1
4 task( 1) is running
5 LockResource task( 1)( 0) R2
5 task( 1) is running
6 task( 1) is running
7 UnlockResource task( 1)( 0) R1
7 task( 1) is running
8 UnlockResource task( 1)( 0) R2
8 task( 1) is running
9 Completion task( 1)( 0) task( 2)( 0) 7 1 0
9 task( 2) is running
10 task( 2) is running
11 LockResource task( 2)( 0) R1
11 task( 2) is running
12 UnlockResource task( 2)( 0) R1
12 task( 2) is running
13 Completion task( 2)( 0) task(63) 13 0 6
17 Preemption task(63) task( 1)( 1)
17 task( 1) is running
18 LockResource task( 1)( 1) R1
18 task( 1) is running
19 LockResource task( 1)( 1) R2
19 task( 1) is running
20 task( 1) is running
21 UnlockResource task( 1)( 1) R1
21 task( 1) is running
22 UnlockResource task( 1)( 1) R2
22 task( 1) is running
23 Completion task( 1)( 1) task( 2)( 1) 6 0 0
23 task( 2) is running
24 LockResource task( 2)( 1) R2
24 task( 2) is running
25 task( 2) is running
26 UnlockResource task( 2)( 1) R2
26 task( 2) is running
27 task( 2) is running
28 LockResource task( 2)( 1) R1
28 task( 2) is running
29 UnlockResource task( 2)( 1) R1
29 task( 2) is running
30 Completion task( 2)( 1) task(63) 10 0 3

```

The output results of **Example 2**:



```

0 task( 3) is running
1 LockResource task( 3)( 0) R2
1 task( 3) is running
2 task( 3) is running
3 UnlockResource task( 3)( 0) R2
3 task( 3) is running
4 Completion task( 3)( 0) task( 1)( 0) 4 0 0
4 task( 1) is running
5 LockResource task( 1)( 0) R1
5 task( 1) is running
6 task( 1) is running
7 task( 1) is running
8 task( 1) is running
9 task( 1) is running
10 UnlockResource task( 1)( 0) R1
10 Preemption task( 1)( 0) task( 2)( 0)
10 task( 2) is running
11 task( 2) is running
12 task( 2) is running
13 task( 2) is running
14 task( 2) is running
15 Completion task( 2)( 0) task( 1)( 0) 7 2 0
15 task( 1) is running
16 task( 1) is running
17 Completion task( 1)( 0) task( 63) 16 0 8
20 Preemption task( 63) task( 3)( 1)
20 task( 3) is running
21 LockResource task( 3)( 1) R2
21 task( 3) is running
22 task( 3) is running
23 UnlockResource task( 3)( 1) R2
23 task( 3) is running
24 Completion task( 3)( 1) task( 63) 4 0 0

```

[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) }

**Example Task Set 1 = { task₁ (1, 2, 6, 15, 1, 4, 2, 5),
task₂ (2, 0, 7, 20, 5, 6, 1, 3) }**

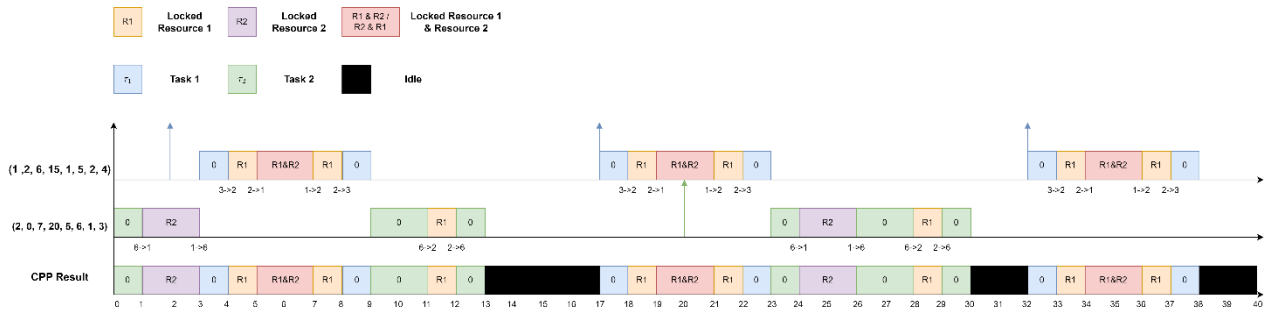
**Example Task Set 2 = { task₁ (1, 1, 8, 31, 1, 3, 5, 7),
task₂ (2, 12, 5, 28, 0, 0, 0, 0),
task₃ (3, 0, 6, 20, 0, 0, 1, 3) }**

Evaluation:

The output format:

Tick	Event	CurrentTask ID	NextTask ID	Response Time	Blocking Time	Preemption Time	Resource Name	Priority Ceiling
##	Preemption	task(ID)(job number)	task(ID)(job number)					
##	Completion	task(ID)(job number)	task(ID)(job number)	##	##	##		
##	LockResource	task(ID)(job number)					R#	## to ##
##	UnlockResource	task(ID)(job number)					R#	## to ##

The output results of **Example 1**:

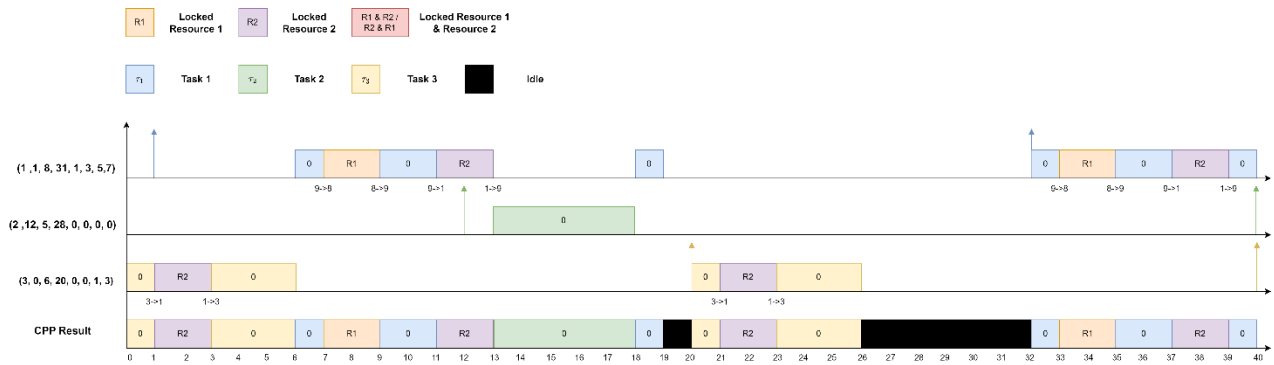


```

0 task( 2) is running
1 LockResource task( 2)( 0) R2 6 to 1
1 task( 2) is running
2 task( 2) is running
3 UnlockResource task( 2)( 0) R2 1 to 6
3 Preemption task( 2)( 0) task( 1)( 0)
3 task( 1) is running
4 LockResource task( 1)( 0) R1 3 to 2
4 task( 1) is running
5 LockResource task( 1)( 0) R2 2 to 1
5 task( 1) is running
6 task( 1) is running
7 UnlockResource task( 1)( 0) R2 1 to 2
7 task( 1) is running
8 UnlockResource task( 1)( 0) R1 2 to 3
8 task( 1) is running
9 Completion task( 1)( 0) task( 2)( 0) 7 1 0
9 task( 2) is running
10 task( 2) is running
11 LockResource task( 2)( 0) R1 6 to 2
11 task( 2) is running
12 UnlockResource task( 2)( 0) R1 2 to 6
12 task( 2) is running
13 Completion task( 2)( 0) task(63) 13 0 6
17 Preemption task(63) task( 1)( 1)
17 task( 1) is running
18 LockResource task( 1)( 1) R1 3 to 2
18 task( 1) is running
19 LockResource task( 1)( 1) R2 2 to 1
19 task( 1) is running
20 task( 1) is running
21 UnlockResource task( 1)( 1) R2 1 to 2
21 task( 1) is running
22 UnlockResource task( 1)( 1) R1 2 to 3
22 task( 1) is running
23 Completion task( 1)( 1) task( 2)( 1) 6 0 0
23 task( 2) is running
24 LockResource task( 2)( 1) R2 6 to 1
24 task( 2) is running
25 task( 2) is running
26 UnlockResource task( 2)( 1) R2 1 to 6
26 task( 2) is running
27 task( 2) is running
28 LockResource task( 2)( 1) R1 6 to 2
28 task( 2) is running
29 UnlockResource task( 2)( 1) R1 2 to 6
29 task( 2) is running
30 Completion task( 2)( 1) task(63) 10 0 3

```

The output results of **Example 2**:



Priority & Resource Ceiling assign:

Task Priority = **Multiples of 3** (i.e. 3, 6, 9.....)

Resource Index : **R1 = 1, R2 = 2**

Resource ceiling = **The highest priority of all tasks that require R - Resource Index**

Example :

Task Set 2 = { task₁ (1, 1, 8, 31, **1**, **3**, 5, 7),
task₂ (2, 12, 5, 28, 0, 0, 0, 0),
task₃ (3, 0, 6, 20, 0, 0, **1**, **3**)}

task₁ period = **31**, task₂ period = **28**, task₃ period = **20**

task₁ priority = **9**, task₂ priority = **6**, task₃ priority = **3 (RM)**

※ task 1's priority is the lowest meanwhile task 3's priority is the highest

The **highest priority** of all tasks that require **R1** = **task1's** priority

The **highest priority** of all tasks that require **R2** = **task3's** priority

R1 ceiling = **task1's** priority – R1 index = 9 – 1 = **8**

R2 ceiling = **task3's** priority – R2 index = 3 – 2 = **1**

Credit:

[PART I] NPCS Implementation [45%]

- The correctness of schedule results of examples. Note the testing task set might not be the same as the given example task set. (35%)
- A report that describes your implementation (please attach the screenshot of the code and **MARK** the modified part). (10%)

[PART II] CPP Implementation [45%]

- The correctness of schedule results of examples. Note the testing task set might not be the same as the given example task set. (35%)
- A report that describes your implementation (please attach the screenshot of the code and **MARK** the modified part). (10%)

[PART III] Performance Analysis [10%]

- Explain the difference between NPCS and CPP, and how each mechanism avoids the deadlock problem.
 - Describe how it works 、 Advantage 、 Disadvantage

※ You must modify the source code.

※ You can set the ticks per second in order to run the project quickly. (in os_cfg.h)

```
#define OS_TICKS_PER_SEC 5u
```

※ 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 must define the algorithm in ucos_ii.h.

```
#define RM 0
```

```
#define FIFO 1
```

```
#define EDF 2
```

```
#define ALGORITHM RM
```

```
#define NPCS 0
```

```
#define CPP 1
```

```
#define PROTOCOL NPCS/_CPP
```

Project submit:

Submit to Moodle

Submit deadline: Dec. 11, 2025 (Thursday) 8:00

File name format: RTOS_Myyyddxxx_PA3.zip

RTOS_Myyyddxxx_PA3.zip includes:

- The report (RTOS_Myyyddxxx_PA3.pdf).
- Folder with the executable μ C/OS-II project (RTOS_Myyyddxxx_PA3).

✂ Plagiarizing is strictly prohibited.

Hints:

1. We also declare shared resources, as follows:

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

2. 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);
```

3. 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;  
    }  
}
```

4. File Tree of PA3:

```

D: .
  RTOS_ Myyyddxxx_PA3.pdf
  RTOS_Myyyddxxx_PA3
    ReadMe.txt
    Micrium
      Software
        uC-CPU
          cpu_cache.h
          cpu_core.c
          cpu_core.h
          cpu_def.h
        Win32
          Visual_Studio
            cpu.h
            cpu_c.c
        uC-LIB
          lib_ascii.c
          lib_ascii.h
          lib_def.h
          lib_math.c
          lib_math.h
          lib_mem.c
          lib_mem.h
          lib_str.c
          lib_str.h
        uCOS-II
          Ports
            Win32
              Visual Studio
                os_cpu.h
                os_cpu_c.c
                os_cpu_c.c.bak
          Source
            os.h
            os_cfg_r.h
            os_core.c
            os_core.c.bak
            os_dbg_r.c
            os_flag.c
            os_mbox.c
            os_mem.c
            os_mutex.c
            os_q.c
            os_sem.c
            os_task.c

```

```

  os_time.c
  os_tmr.c
  os_trace.h
  ucos_ii.c
  ucos_ii.h
  ucos_ii.h.bak
  Microsoft
    BSP
      Windows
        bsp_cpu.c
    Windows
      Kernel
        app_cfg.h
        cpu_cfg.h
        lib_cfg.h
      OS2
        app_hooks.c
        app_hooks.c.bak
        main.c
        main.c.bak
        os_cfg.h
        os_cfg.h.bak
      VS
        OS2.sln
        OS2.vcxproj
        OS2.vcxproj.filters
        OS2.vcxproj.user
        Output.txt
        TaskSet.txt

```