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:

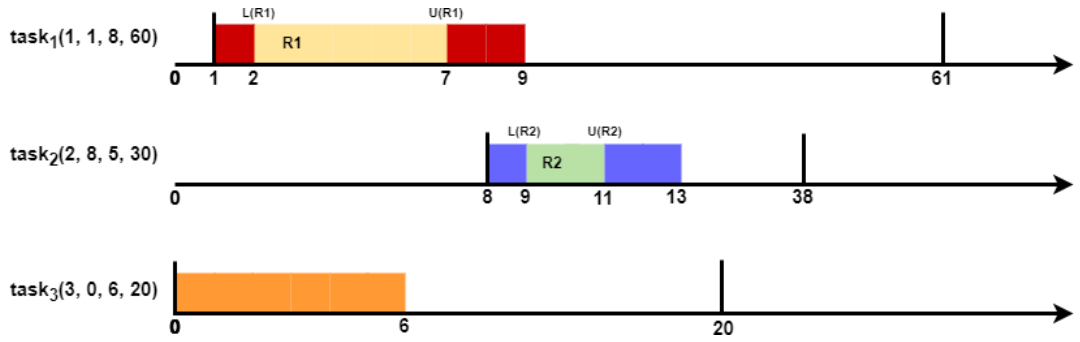
uC/OS-II uses a variation of the priority inheritance protocol to deal with priority inversions. In this assignment, you are going to implement the NPCS based on the RM scheduler in uC/OS-II.

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

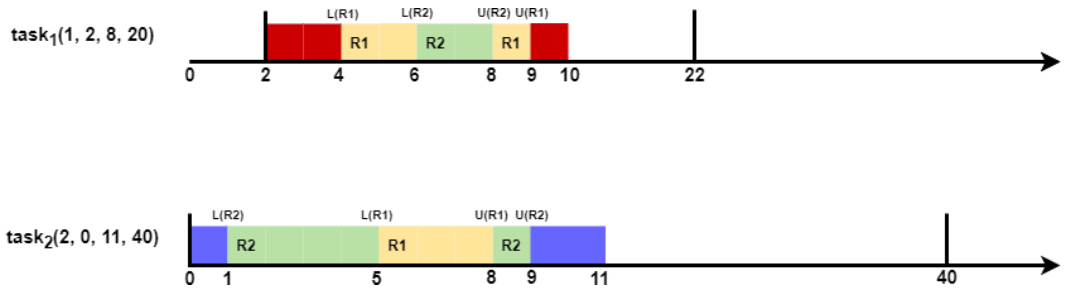
Periodic Task Set = { $task_{ID}$ (ID, arrival time, execution time, period, R1 lock, R1 unlock, R2 lock, R2 unlock) }

※ 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 correctness of schedule results of examples. Note the testing task set might not be thesame as the given example task set.

EX 1：

Tick	Event	CurrentTask ID	NextTask ID
6	Completion	task(3)(0)	task(1)(0)
7	Task1 get R1		
12	Task1 release R1		
12	Preemption	task(1)(0)	task(2)(0)
13	Task2 get R2		
15	Task2 release R2		
17	Completion	task(2)(0)	task(1)(0)
19	Completion	task(1)(0)	task(63)
20	Preemption	task(63)	task(3)(1)
26	Completion	task(3)(1)	task(63)
38	Preemption	task(63)	task(2)(1)
39	Task2 get R2		
41	Task2 release R2		
41	Preemption	task(2)(1)	task(3)(2)
47	Completion	task(3)(2)	task(2)(1)
49	Completion	task(2)(1)	task(63)
60	Preemption	task(63)	task(3)(3)
66	Completion	task(3)(3)	task(1)(1)
67	Task1 get R1		
72	Task1 release R1		
72	Preemption	task(1)(1)	task(2)(2)
73	Task2 get R2		
75	Task2 release R2		
77	Completion	task(2)(2)	task(1)(1)
79	Completion	task(1)(1)	task(63)
80	Preemption	task(63)	task(3)(4)
86	Completion	task(3)(4)	task(63)
98	Preemption	task(63)	task(2)(3)
99	Task2 get R2		

6	Completion	task(3)(0)	task(1)(0)
7	Task1 get R1		
12	Task1 release R1		
12	Preemption	task(1)(0)	task(2)(0)
13	Task2 get R2		
15	Task2 release R2		
17	Completion	task(2)(0)	task(1)(0)
19	Completion	task(1)(0)	task(63)
20	Preemption	task(63)	task(3)(1)
26	Completion	task(3)(1)	task(63)
38	Preemption	task(63)	task(2)(1)
39	Task2 get R2		
41	Task2 release R2		
41	Preemption	task(2)(1)	task(3)(2)
47	Completion	task(3)(2)	task(2)(1)
49	Completion	task(2)(1)	task(63)
60	Preemption	task(63)	task(3)(3)
66	Completion	task(3)(3)	task(1)(1)
67	Task1 get R1		
72	Task1 release R1		
72	Preemption	task(1)(1)	task(2)(2)
73	Task2 get R2		
75	Task2 release R2		
77	Completion	task(2)(2)	task(1)(1)
79	Completion	task(1)(1)	task(63)
80	Preemption	task(63)	task(3)(4)
86	Completion	task(3)(4)	task(63)
98	Preemption	task(63)	task(2)(3)
99	Task2 get R2		

EX 2：

Tick	Event	CurrentTask ID	NextTask ID
1	Task2 get R2		
5	Task2 get R1		
8	Task2 release R1		
9	Task2 release R2		
9	Preemption	task(2)(0)	task(1)(0)
11	Task1 get R1		
13	Task1 get R2		
15	Task1 release R2		
16	Task1 release R1		
17	Completion	task(1)(0)	task(2)(0)
19	Completion	task(2)(0)	task(63)
22	Preemption	task(63)	task(1)(1)
24	Task1 get R1		
26	Task1 get R2		
28	Task1 release R2		
29	Task1 release R1		
30	Completion	task(1)(1)	task(63)
40	Preemption	task(63)	task(2)(1)
41	Task2 get R2		
45	Task2 get R1		
48	Task2 release R1		
49	Task2 release R2		
49	Preemption	task(2)(1)	task(1)(2)
51	Task1 get R1		
53	Task1 get R2		
55	Task1 release R2		
56	Task1 release R1		
57	Completion	task(1)(2)	task(2)(1)
59	Completion	task(2)(1)	task(63)
62	Preemption	task(63)	task(1)(3)
64	Task1 get R1		
66	Task1 get R2		
68	Task1 release R2		
69	Task1 release R1		
70	Completion	task(1)(3)	task(63)
80	Preemption	task(63)	task(2)(2)
81	Task2 get R2		
85	Task2 get R1		
88	Task2 release R1		
89	Task2 release R2		
89	Preemption	task(2)(2)	task(1)(4)
91	Task1 get R1		
93	Task1 get R2		
95	Task1 release R2		
96	Task1 release R1		
97	Completion	task(1)(4)	task(2)(2)
99	Completion	task(2)(2)	task(63)

```
 1    Task2 get R2
 5    Task2 get R1
 8    Task2 release R1
 9    Task2 release R2
 9    Preemption   task( 2)( 0)    task( 1)( 0)
11    Task1 get R1
13    Task1 get R2
15    Task1 release R2
16    Task1 release R1
17    Completion   task( 1)( 0)    task( 2)( 0)
19    Completion   task( 2)( 0)    task(63)
22    Preemption   task(63)        task( 1)( 1)
24    Task1 get R1
26    Task1 get R2
28    Task1 release R2
29    Task1 release R1
30    Completion   task( 1)( 1)    task(63)
40    Preemption   task(63)        task( 2)( 1)
41    Task2 get R2
45    Task2 get R1
48    Task2 release R1
49    Task2 release R2
49    Preemption   task( 2)( 1)    task( 1)( 2)
51    Task1 get R1
53    Task1 get R2
55    Task1 release R2
56    Task1 release R1
57    Completion   task( 1)( 2)    task( 2)( 1)
59    Completion   task( 2)( 1)    task(63)
62    Preemption   task(63)        task( 1)( 3)
64    Task1 get R1
66    Task1 get R2
68    Task1 release R2
69    Task1 release R1
70    Completion   task( 1)( 3)    task(63)
80    Preemption   task(63)        task( 2)( 2)
81    Task2 get R2
85    Task2 get R1
88    Task2 release R1
89    Task2 release R2
89    Preemption   task( 2)( 2)    task( 1)( 4)
91    Task1 get R1
93    Task1 get R2
95    Task1 release R2
96    Task1 release R1
97    Completion   task( 1)( 4)    task( 2)( 2)
99    Completion   task( 2)( 2)    task(63)
```

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

最一開始先新建兩個 event 去個別代表資源 R1 及資源 R2，而宣告位子必須在 main.c 中為global variable 的型態，

```
/*
*********************************************************************************************************
*                                          LOCAL GLOBAL VARIABLES
*********************************************************************************************************
*/

#define TASK_STACKSIZE       2048
static  OS_STK  StartupTaskStk[APP_CFG_STARTUP_TASK_STK_SIZE];

//AddedCodePA3
OS_EVENT* R1;
OS_EVENT* R2;
```

而初始化若是太早可能造成程式有問題，因此要在 TCB、 task 等都初始並創建完後再初始化 event R1 及 R2 才行，且 R1 和 R2 是不同資源，因此不能採用 R=OSSemCreate(2)去設定：

```
int  main (void)
{
#if OS_TASK_NAME_EN > 0u
    CPU_INT08U  os_err;
#endif

    CPU_IntInit();

    Mem_Init();                                          /* Initialize Memory Managment Module                   */
    CPU_IntDis();                                        /* Disable all Interrupts                               */
    CPU_Init();                                          /* Initialize the uC/CPU services                       */

    OSInit();                                            /* Initialize uC/OS-II                                  */
    /*Initialize Output File*/
    OutFileInit();

    /*Input File*/
    InputFile();

    /*Dynamic Create the Stack size*/
    Task_STK = malloc(TASK_NUMBER * sizeof(int*));

    /* for each pointer, allocate storage for an array of ints */
    int n;
    for (n = 0; n < TASK_NUMBER; n++) {
        Task_STK[n] = malloc(TASK_STACKSIZE * sizeof(int));
        OSTaskCreateExt(task2,                    /*Create the task2*/
            &TaskParameter[n],
            &Task_STK[n][TASK_STACKSIZE - 1],
            TaskParameter[n].TaskPriority,
            TaskParameter[n].TaskID,
            &Task_STK[n][0],
            TASK_STACKSIZE,
            &TaskParameter[n],
            (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));
    }

    //AddedCodePA3
    INT8U err;
    R1 = OSSemCreate(1);
    R2 = OSSemCreate(1);
```

因為是想透過Semaphore去達成搶到資源時，不讓其他task搶走，因此需要修改main.c下的task，使其不單單只有做delay的功能：

```
283
284 void task2(void* p_arg) {
285     task_para_set* task_data;
286     task_data = p_arg;
287     while (1)
288     {
289         INT8U err;
290
291         while (1) {
292
293             if ((OSTimeGet() == (OSTCBCur->OSTCBCyclesStart + OSTCBCur->OSTCBR2Lock - 1)) && OSTCBCur->OSTCBR2Lock != 0) {
294                 OSSemPend(R2, 0, &err);
295             }
296             else if ((OSTimeGet() == (OSTCBCur->OSTCBCyclesStart + OSTCBCur->OSTCBR1Lock - 1)) && OSTCBCur->OSTCBR1Lock != 0) {
297                 OSSemPend(R1, 0, &err);
298             }
299             else if ((OSTimeGet() == (OSTCBCur->OSTCBCyclesStart + OSTCBCur->OSTCBR1Unlock - 2)) && OSTCBCur->OSTCBR1Unlock != 0) {
300                 OSSemPost(R1);
301             }
302             else if ((OSTimeGet() == (OSTCBCur->OSTCBCyclesStart + OSTCBCur->OSTCBR2Unlock - 2)) && OSTCBCur->OSTCBR2Unlock != 0) {
303                 OSSemPost(R2);
304             }
305
306             if (OSTCBCur->OSTCBCyclesCount == OSTCBCur->OSTCBCyclesExecution) {
307                 OSTCBCur->OSTCBCyclesCount = 0;
308                 OSTimeDly(OSTCBCur->OSTCBDly);
309             }
310         }
311     }
312 }
313
```

此外，在 ucos_ii.h 中宣告所需要用到的讀檔參數以及 R1、R2 的 Lock 與 Unlock 相對時間，以便後續進行是否執行 semaphore 的判斷：

```
92  typedef struct task_para_set {
93      INT16U TaskID;
94      INT16U TaskArriveTime;
95      INT16U TaskExecutionTime;
96      INT16U TaskPeriodic;
97      INT16U TaskNumber;
98      INT16U TaskPriority;
99      //AddedCodePA3
100     INT16U TaskR1Lock;
101     INT16U TaskR1Unlock;
102     INT16U TaskR2Lock;
103     INT16U TaskR2Unlock;
104 } task_para_set;
105
```

其中下圖中的OSTCBUsingResource表示該 task 是否正在使用資源，0為否1為是：

```
678     //AddedCodePA1part2
679     INT32U          OSTCBCyclesExecution;  /* Setting about Execution Time */
680     INT32U          OSTCBCyclesCount;      /* Count Cycles */
681     INT32U          OSTCBCyclesArrive;     /* When arrive? */
682     INT32U          OSTCBCyclesEnd;        /* When cycle end? */
683     INT32U          OSTCBJobNumber;        /* Executing the N-th Job */
684     INT32U          OSTCBCyclesPeriod;     /* Period */
685     INT32U          OSTCBCyclesSwitchStart;/* To know #swich the cycle start */
686     INT32U          OSTCBMyTaskCtxTimes;   /* The task's ctx times */
687     //AddedCodePA3part1
688     INT32U          OSTCBR1Lock;           /* The relative time of R1 Lock */
689     INT32U          OSTCBR1Unlock;         /* The relative time of R1 Unlock */
690     INT32U          OSTCBR2Lock;           /* The relative time of R2 Lock */
691     INT32U          OSTCBR2Unlock;         /* The relative time of R2 Unlock */
692     INT32U          OSTCBUsingResource;    /* Check using resource or not */
693     INT32U          OSTCBOriPrio;          /* Original Priority */
694     INT32U          OSTCBDeadline;         /* Deadline of the task */
695 #endif
696
```

而在參數宣告完之後，需在OS_TCBInit 中進行初始化，也必須在此新增讀檔內容並存取資源 R1 及資源 R2 的 Unlock 和 Lock 時間：

```
2273        char str[MAX];
2274        char* ptr;
2275        char* pTmp = NULL;
2276        int TaskInfo[INFO], k, 1 = 0;
2277        TASK_NUMBER = 0;
2278        while (!feof(fp) && prio != 63)
2279        {
2280            k = 0;
2281            memset(str, 0, sizeof(str));
2282            fgets(str, sizeof(str) - 1, fp);
2283            ptr = strtok_s(str, " ", &pTmp);
2284            while (ptr != NULL)
2285            {
2286                TaskInfo[k] = atoi(ptr);
2287                ptr = strtok_s(NULL, " ", &pTmp);
2288                /*printf("Info: %d\n", task_inf[i]);*/
2289                if (k == 0) {
2290                    TASK_NUMBER++;
2291                    TaskParameter[1].TaskID = TASK_NUMBER;
2292                }
2293                else if (k == 1) {
2294                    TaskParameter[1].TaskArriveTime = TaskInfo[k];
2295                }
2296                else if (k == 2) {
2297                    TaskParameter[1].TaskExecutionTime = TaskInfo[k];
2298                }
2299                else if (k == 3) {
2300                    TaskParameter[1].TaskPeriodic = TaskInfo[k];
2301                    TaskParameter[1].TaskPriority = TaskInfo[k]; //Initial Priority=Period
2302                }
2303                //AddedCodePA3
2304                else if (k == 4) {
2305                    TaskParameter[1].TaskR1Lock = TaskInfo[k];
2306                }
2307                else if (k == 5) {
2308                    TaskParameter[1].TaskR1Unlock = TaskInfo[k];
2309                }
2310                else if (k == 6) {
2311                    TaskParameter[1].TaskR2Lock = TaskInfo[k];
2312                }
2313                else if (k == 7) {
2314                    TaskParameter[1].TaskR2Unlock = TaskInfo[k];
2315                }
2316                k++;
2317            }
2318            1++;
2319        }
2320        fclose(fp);
2321        if (prio != 63) {
2322            unsigned int delay = TaskParameter[id - 1].TaskArriveTime;
2323            unsigned int exetime = TaskParameter[id - 1].TaskExecutionTime;
2324            ptcb->OSTCBCyclesExecution = exetime;
2325            ptcb->OSTCBCyclesCount = 0u;
2326            ptcb->OSTCBJobNumber = 0u;
2327            ptcb->OSTCBCyclesEnd = 0u;
2328            ptcb->OSTCBCyclesSwitchStart = 0u;
2329            ptcb->OSTCBMyTaskCtxTimes = 0u;
2330            ptcb->OSTCBCyclesPeriod = TaskParameter[id - 1].TaskPeriodic;
2331            ptcb->OSTCBCyclesArrive = TaskParameter[id - 1].TaskArriveTime;
2332            //TryingPA3
2333            ptcb->OSTCBR1Lock = TaskParameter[id - 1].TaskR1Lock;
2334            ptcb->OSTCBR1Unlock = TaskParameter[id - 1].TaskR1Unlock;
2335            ptcb->OSTCBR2Lock = TaskParameter[id - 1].TaskR2Lock;
2336            ptcb->OSTCBR2Unlock = TaskParameter[id - 1].TaskR2Unlock;
2337            ptcb->OSTCBUsingResource = 0;
2338            ptcb->OSTCBDeadline = ptcb->OSTCBCyclesArrive + TaskParameter[id - 1].TaskPeriodic;
2339            while (ptcb->OSTCBDly != delay) {
2340                ptcb->OSTCBDly++;
2341            }
2342        }
2343        ////////////////////
```

接下來必須在 os_sem.c檔案 中去修改 OSSemPend 及 OSSemPost 以符合所要的打印格式。在 OSSemPend 中，因為 while 迴圈會在同一 tick 多次執行，因此會一直進入 OSSemPend 程式中，為了避免瘋狂打印的情形發生，要注意的是有一數值為 OSEventCnt 在第一次進入時為1之後變成0，可以用這個數值判斷在第一次進來時打印即可，後續再進來就不用繼續執行這些動作直接 return，除此之外，在取得資源時就將 OSTCBUsingResource 進行 +1，代表這個任務使用了一個資源，因為有兩個資源的關係，因此最多OSTCBUsingResource 會加到2：

```
323  void  OSSemPend (OS_EVENT  *pevent,
324                   INT32U     timeout,
325                   INT8U     *perr)
326  {
327  #if OS_CRITICAL_METHOD == 3u                       /* Allocate storage for CPU status register      */
328      OS_CPU_SR  cpu_sr = 0u;
329  #endif
330
331
332  #ifdef OS_SAFETY_CRITICAL
333      if (perr == (INT8U *)0) {
334          OS_SAFETY_CRITICAL_EXCEPTION();
335          return;
336      }
337  #endif
338
339  #if OS_ARG_CHK_EN > 0u
340      if (pevent == (OS_EVENT *)0) {                 /* Validate 'pevent'                             */
341          *perr = OS_ERR_PEVENT_NULL;
342          return;
343      }
344  #endif
345
346      OS_TRACE_SEM_PEND_ENTER(pevent, timeout);
347
348      if (pevent->OSEventCnt == 0)//AddedCodePA3
349          return;
350
351      if (pevent->OSEventType != OS_EVENT_TYPE_SEM) {   /* Validate event block type                     */
352          *perr = OS_ERR_EVENT_TYPE;
353          OS_TRACE_SEM_PEND_EXIT(*perr);
354          return;
355      }
356      if (OSIntNesting > 0u) {                       /* See if called from ISR ...                    */
357          *perr = OS_ERR_PEND_ISR;                   /* ... can't PEND from an ISR                    */
358          OS_TRACE_SEM_PEND_EXIT(*perr);
359          return;
360      }
361      if (OSLockNesting > 0u) {                      /* See if called with scheduler locked ...       */
362          *perr = OS_ERR_PEND_LOCKED;                /* ... can't PEND when locked                    */
363          OS_TRACE_SEM_PEND_EXIT(*perr);
364          return;
365      }
366      OS_ENTER_CRITICAL();
367      if (pevent->OSEventCnt > 0u) {                 /* If sem. is positive, resource available ...   */
368          pevent->OSEventCnt--;                      /* ... decrement semaphore only if positive.     */
369          OS_EXIT_CRITICAL();
370          *perr = OS_ERR_NONE;
371          OS_TRACE_SEM_PEND_EXIT(*perr);
372
373          //AddedCodePA3
374          if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
375              if ((OSTimeGet() == (OSTCBCur->OSTCBCyclesStart + OSTCBCur->OSTCBR2Lock - 1)) && OSTCBCur->OSTCBR2Lock != 0) {
376                  printf("%2d\tTask%d get R2\n", OSTimeGet(), OSTCBCur->OSTCBId);
377                  fprintf(Output_fp, "%2d\tTask%d get R2\n", OSTimeGet(), OSTCBCur->OSTCBId);
378                  OSTCBCur->OSTCBUsingResource ++;
379                  //printf("%d\n", OSTCBCur->OSTCBUsingResource);
380              }
381              else if ((OSTimeGet() == (OSTCBCur->OSTCBCyclesStart + OSTCBCur->OSTCBR1Lock - 1)) && OSTCBCur->OSTCBR1Lock != 0) {
382                  printf("%2d\tTask%d get R1\n", OSTimeGet(), OSTCBCur->OSTCBId);
383                  fprintf(Output_fp, "%2d\tTask%d get R1\n", OSTimeGet(), OSTCBCur->OSTCBId);
384                  OSTCBCur->OSTCBUsingResource ++;
385                  //printf("%d\n", OSTCBCur->OSTCBUsingResource);
386              }
387          }fclose(Output_fp);
388          return;
389      }
```

在 OSSemPost 中設計概念與OSSemPend 類似，為了避免 task 下的 while 迴圈在同一 tick 下多次進入，也會撰寫 if 判斷式去檢查是否為第一次進入，並在確定 Pend 時將 OSTCBUsingResource 進行 -1 的動作代表已釋放一個資源：

```
INT8U  OSSemPost (OS_EVENT *pevent)
{
#if OS_CRITICAL_METHOD == 3u                       /* Allocate storage for CPU status register      */
    OS_CPU_SR  cpu_sr = 0u;
#endif


#if OS_ARG_CHK_EN > 0u
    if (pevent == (OS_EVENT *)0) {                 /* Validate 'pevent'                             */
        return (OS_ERR_PEVENT_NULL);
    }
#endif

    OS_TRACE_SEM_POST_ENTER(pevent);

    if (pevent->OSEventCnt == 1)//AddedCodePA3
        return;

    if (pevent->OSEventType != OS_EVENT_TYPE_SEM) {   /* Validate event block type                  */
        OS_TRACE_SEM_POST_EXIT(OS_ERR_EVENT_TYPE);
        return (OS_ERR_EVENT_TYPE);
    }
    OS_ENTER_CRITICAL();
    if (pevent->OSEventGrp != 0u) {                /* See if any task waiting for semaphore         */
                                                   /* Ready HPT waiting on event                    */
        (void)OS_EventTaskRdy(pevent, (void *)0, OS_STAT_SEM, OS_STAT_PEND_OK);
        OS_EXIT_CRITICAL();
        OS_Sched();                                /* Find HPT ready to run                         */
        OS_TRACE_SEM_POST_EXIT(OS_ERR_NONE);
        return (OS_ERR_NONE);
    }
    if (pevent->OSEventCnt < 65535u) {             /* Make sure semaphore will not overflow         */
        pevent->OSEventCnt++;                      /* Increment semaphore count to register event   */
        OS_EXIT_CRITICAL();
        OS_TRACE_SEM_POST_EXIT(OS_ERR_NONE);
        //AddedCodePA3
        if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
            if ((OSTimeGet() == (OSTCBCur->OSTCBCyclesStart + OSTCBCur->OSTCBR1Unlock - 2)) && OSTCBCur->OSTCBR1Unlock != 0) {
                printf("%2d\tTask%d release R1\n", OSTimeGet() + 1, OSTCBCur->OSTCBId);
                fprintf(Output_fp, "%2d\tTask%d release R1\n", OSTimeGet() + 1, OSTCBCur->OSTCBId);
                OSTCBCur->OSTCBUsingResource --;
            }
            else if ((OSTimeGet() == (OSTCBCur->OSTCBCyclesStart + OSTCBCur->OSTCBR2Unlock - 2)) && OSTCBCur->OSTCBR2Unlock != 0) {
                printf("%2d\tTask%d release R2\n", OSTimeGet() + 1, OSTCBCur->OSTCBId);
                fprintf(Output_fp, "%2d\tTask%d release R2\n", OSTimeGet() + 1, OSTCBCur->OSTCBId);
                OSTCBCur->OSTCBUsingResource --;
            }fclose(Output_fp);
        }
        return (OS_ERR_NONE);
    }
    OS_EXIT_CRITICAL();                            /* Semaphore value has reached its maximum       */
    OS_TRACE_SEM_POST_EXIT(OS_ERR_SEM_OVF);

    return (OS_ERR_SEM_OVF);
}
```

因為在 Non-preemptive critical section, NPCS 中，哪個任務搶到資源則就要等該任務將資源釋放以後才能換其他任務去使用，因此在 OSIntExit 中需要新增一判斷，當 OSTCBCur->OSTCBUsingResource != 0 時代表正在使用資源，直接跳出 OSIntExit，避免正在使用資源的 task 被其他 task 打斷執行：

```
699  void  OSIntExit (void)
700  {
701  #if OS_CRITICAL_METHOD == 3u                          /* Allocate storage for CPU status register */
702      OS_CPU_SR  cpu_sr = 0u;
703  #endif
704      if(OSTimeGet() == 100)
705          OSTimeGet();
706
707      if (OSRunning == OS_TRUE) {
708          OS_ENTER_CRITICAL();
709          if (OSIntNesting > 0u) {                      /* Prevent OSIntNesting from wrapping       */
710              OSIntNesting--;
711          }
712          if (OSIntNesting == 0u && OSTCBCur->OSTCBCyclesTot != OSTCBCur->OSTCBCyclesExecution && OSTCBCur->OSTCBUsingResource==0) {
713              if (OSLockNesting == 0u) {                /* ... and not locked.                      */
714                  OS_SchedNew();
715                  OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
716                  if (OSPrioHighRdy != OSPrioCur) {     /* No Ctx Sw if current task is highest rdy */
717  #if OS_TASK_PROFILE_EN > 0u
718                      OSTCBHighRdy->OSTCBCtxSwCtr++;    /* Inc. # of context switches to this task  */
719  #endif
720                      OSCtxSwCtr++;                     /* Keep track of the number of ctx switches */
721
722                      //AddedCodePAlpart2
723                      OSTCBCur->OSTCBMyTaskCtxTimes++;
724                      OSTCBHighRdy->OSTCBMyTaskCtxTimes++;
```

[PART II] CPP Implementation

Objective:

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

Problem Definition:

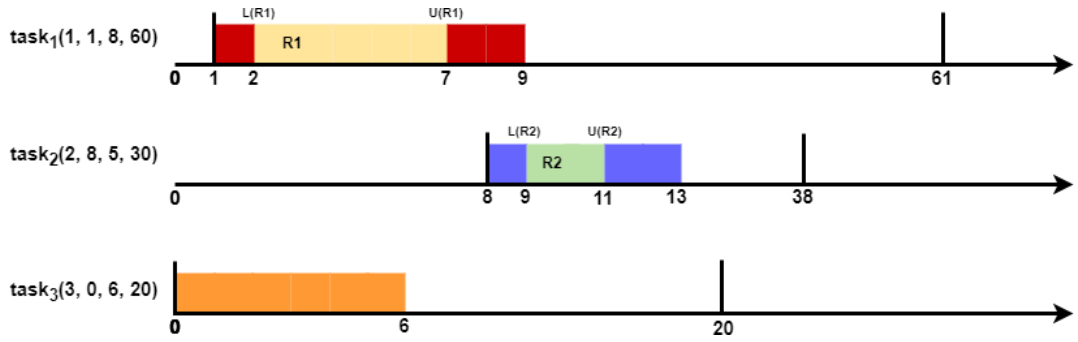
uC/OS-II uses a variation of the priority inheritance protocol to deal with priority inversions. In this assignment, you are going to implement the CPP based on the RM scheduler in uC/OS-II.

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

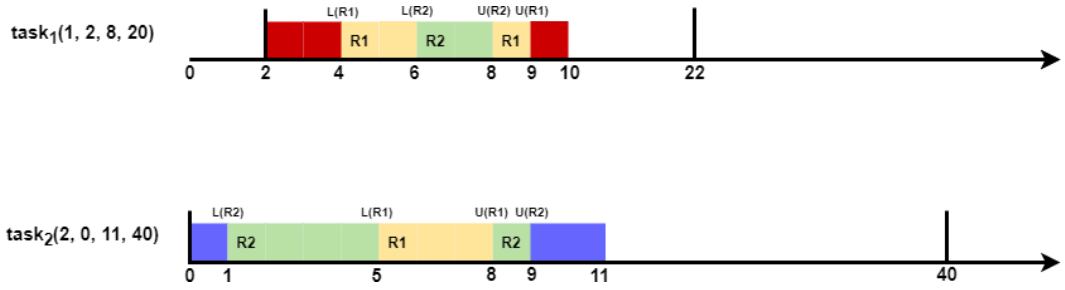
Periodic Task Set = { $task_{ID}$ (ID, arrival time, execution time, period, R1 lock, R1 unlock, R2 lock, R2 unlock) }

※ 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)}



The correctness of schedule results of examples. Note the testing task set might not be the same as the given example task set.

EX 1：

```
Tick    Event          CurrentTask ID    NextTask ID
 6      Completion       task( 3)( 0)    task( 1)( 0)
 7      task1 get R1                                          12 to 11
 8      Preemption       task( 1)( 0)    task( 2)( 0)
 9      task2 get R2                                           8 to 7
11      task2 release R2                                    7 to 8
13      Completion       task( 2)( 0)    task( 1)( 0)
17      task1 release R1                                   11 to 12
19      Completion       task( 1)( 0)    task(63)
20      Preemption       task(63)        task( 3)( 1)
26      Completion       task( 3)( 1)    task(63)
38      Preemption       task(63)        task( 2)( 1)
39      task2 get R2                                           8 to 7
40      Preemption       task( 2)( 1)    task( 3)( 2)
46      Completion       task( 3)( 2)    task( 2)( 1)
47      task2 release R2                                    7 to 8
49      Completion       task( 2)( 1)    task(63)
60      Preemption       task(63)        task( 3)( 3)
66      Completion       task( 3)( 3)    task( 1)( 1)
67      task1 get R1                                          12 to 11
68      Preemption       task( 1)( 1)    task( 2)( 2)
69      task2 get R2                                           8 to 7
71      task2 release R2                                    7 to 8
73      Completion       task( 2)( 2)    task( 1)( 1)
77      task1 release R1                                   11 to 12
79      Completion       task( 1)( 1)    task(63)
80      Preemption       task(63)        task( 3)( 4)
86      Completion       task( 3)( 4)    task(63)
98      Preemption       task(63)        task( 2)( 3)
99      task2 get R2                                           8 to 7
100     Preemption       task( 2)( 3)    task( 3)( 5)
```

```
 6    Completion  task( 3)( 0)    task( 1)( 0)
 7    task1 get R1                                  12 to 11
 8    Preemption  task( 1)( 0)    task( 2)( 0)
 9    task2 get R2                                   8 to 7
11    task2 release R2                            7 to 8
13    Completion  task( 2)( 0)    task( 1)( 0)
17    task1 release R1                           11 to 12
19    Completion  task( 1)( 0)    task(63)
20    Preemption  task(63)        task( 3)( 1)
26    Completion  task( 3)( 1)    task(63)
38    Preemption  task(63)        task( 2)( 1)
39    task2 get R2                                   8 to 7
40    Preemption  task( 2)( 1)    task( 3)( 2)
46    Completion  task( 3)( 2)    task( 2)( 1)
47    task2 release R2                            7 to 8
49    Completion  task( 2)( 1)    task(63)
60    Preemption  task(63)        task( 3)( 3)
66    Completion  task( 3)( 3)    task( 1)( 1)
67    task1 get R1                                  12 to 11
68    Preemption  task( 1)( 1)    task( 2)( 2)
69    task2 get R2                                   8 to 7
71    task2 release R2                            7 to 8
73    Completion  task( 2)( 2)    task( 1)( 1)
77    task1 release R1                           11 to 12
79    Completion  task( 1)( 1)    task(63)
80    Preemption  task(63)        task( 3)( 4)
86    Completion  task( 3)( 4)    task(63)
98    Preemption  task(63)        task( 2)( 3)
99    task2 get R2                                   8 to 7
100   Preemption  task( 2)( 3)    task( 3)( 5)
```

EX 2：

```
Tick    Event         CurrentTask ID    NextTask ID
 1      task2 get R2                                        8 to 3
 5      task2 get R1                                        3 to 1
 8      task2 release R1                                1 to 3
 9      task2 release R2                                3 to 8
 9      Preemption       task( 2)( 0)    task( 1)( 0)
11      task1 get R1                                        4 to 1
13      task1 get R2                                        1 to 1
15      task1 release R2                                1 to 1
16      task1 release R1                                1 to 4
17      Completion       task( 1)( 0)    task( 2)( 0)
19      Completion       task( 2)( 0)    task(63)
22      Preemption       task(63)        task( 1)( 1)
24      task1 get R1                                        4 to 1
26      task1 get R2                                        1 to 1
28      task1 release R2                                1 to 1
29      task1 release R1                                1 to 4
30      Completion       task( 1)( 1)    task(63)
40      Preemption       task(63)        task( 2)( 1)
41      task2 get R2                                        8 to 3
45      task2 get R1                                        3 to 1
48      task2 release R1                                1 to 3
49      task2 release R2                                3 to 8
49      Preemption       task( 2)( 1)    task( 1)( 2)
51      task1 get R1                                        4 to 1
53      task1 get R2                                        1 to 1
55      task1 release R2                                1 to 1
56      task1 release R1                                1 to 4
57      Completion       task( 1)( 2)    task( 2)( 1)
59      Completion       task( 2)( 1)    task(63)
62      Preemption       task(63)        task( 1)( 3)
64      task1 get R1                                        4 to 1
66      task1 get R2                                        1 to 1
68      task1 release R2                                1 to 1
69      task1 release R1                                1 to 4
70      Completion       task( 1)( 3)    task(63)
80      Preemption       task(63)        task( 2)( 2)
81      task2 get R2                                        8 to 3
85      task2 get R1                                        3 to 1
88      task2 release R1                                1 to 3
89      task2 release R2                                3 to 8
89      Preemption       task( 2)( 2)    task( 1)( 4)
91      task1 get R1                                        4 to 1
93      task1 get R2                                        1 to 1
95      task1 release R2                                1 to 1
96      task1 release R1                                1 to 4
97      Completion       task( 1)( 4)    task( 2)( 2)
99      Completion       task( 2)( 2)    task(63)
```

```
 1    task2 get R2                                        8 to 3
 5    task2 get R1                                        3 to 1
 8    task2 release R1                                1 to 3
 9    task2 release R2                                3 to 8
 9     Preemption  task( 2)( 0)    task( 1)( 0)
11    task1 get R1                                        4 to 1
13    task1 get R2                                        1 to 1
15    task1 release R2                                1 to 1
16    task1 release R1                                1 to 4
17     Completion  task( 1)( 0)    task( 2)( 0)
19     Completion  task( 2)( 0)    task(63)
22     Preemption  task(63)        task( 1)( 1)
24    task1 get R1                                        4 to 1
26    task1 get R2                                        1 to 1
28    task1 release R2                                1 to 1
29    task1 release R1                                1 to 4
30     Completion  task( 1)( 1)    task(63)
40     Preemption  task(63)        task( 2)( 1)
41    task2 get R2                                        8 to 3
45    task2 get R1                                        3 to 1
48    task2 release R1                                1 to 3
49    task2 release R2                                3 to 8
49     Preemption  task( 2)( 1)    task( 1)( 2)
51    task1 get R1                                        4 to 1
53    task1 get R2                                        1 to 1
55    task1 release R2                                1 to 1
56    task1 release R1                                1 to 4
57     Completion  task( 1)( 2)    task( 2)( 1)
59     Completion  task( 2)( 1)    task(63)
62     Preemption  task(63)        task( 1)( 3)
64    task1 get R1                                        4 to 1
66    task1 get R2                                        1 to 1
68    task1 release R2                                1 to 1
69    task1 release R1                                1 to 4
70     Completion  task( 1)( 3)    task(63)
80     Preemption  task(63)        task( 2)( 2)
81    task2 get R2                                        8 to 3
85    task2 get R1                                        3 to 1
88    task2 release R1                                1 to 3
89    task2 release R2                                3 to 8
89     Preemption  task( 2)( 2)    task( 1)( 4)
91    task1 get R1                                        4 to 1
93    task1 get R2                                        1 to 1
95    task1 release R2                                1 to 1
96    task1 release R1                                1 to 4
97     Completion  task( 1)( 4)    task( 2)( 2)
99     Completion  task( 2)( 2)    task(63)
```

A report that describes your implementation (please attach the screenshot of the code and MARK the modified part).

一開始會先在 ucos_ii.h 宣告新的參數去儲存數值，會宣告全域變數去儲存 R1_PRIO、R2_PRIO，除此之外，撰寫了 Find_R1_R2_PRIO 去尋找 R1 和 R2 的優先權，而同樣的由於 task 執行時在同一秒會一直多次進入OSMutexPend，因此宣告了一個變數 OSEventEnterMutexCnt 在OS_EVENT 之下，去紀錄是否為第一次進入 Mutex。

```
61  *********************************************************************
62  *                                                MISCELLANEOUS
63  *********************************************************************
64  */
65
66
67  /*End time for the simulation*/
68  #define SYSTEM_END_TIME 100
69
70  /*Input File*/
71  FILE* fp;
72  #define INPUT_FILE_NAME "./TaskSet.txt"
73  #define OUTPUT_FILE_NAME "./Output.txt"
74  #define MAX 20              //Task maimum number
75  #define INFO 10              //information of task//AddedCodePA3part1
76
77  //AddedCodePA3part2
78  int R1_PRIO;
79  int R2_PRIO;
80
81  /*Input File*/
82
83  /*Output File*/
84  FILE* Output_fp;
85  errno_t Output_err;
86  /*Output File*/
```

```
1342  *********************************************************************
1343  *                                                MISCELLANEOUS
1344  *********************************************************************
1345  */
1346
1347  void          OSInit                (void);
1348
1349  void          OSIntEnter            (void);
1350  void          OSIntExit             (void);
1351
1352  #ifdef OS_SAFETY_CRITICAL_IEC61508
1353  void          OSSafetyCriticalStart (void);
1354  #endif
1355
1356  #if OS_SCHED_LOCK_EN > 0u
1357  void          OSSchedLock           (void);
1358  void          OSSchedUnlock         (void);
1359  #endif
1360
1361  void          OSStart               (void);
1362
1363  void          OSStatInit            (void);
1364
1365  INT16U        OSVersion             (void);
1366
1367  //AddedCodePA3part2
1368  void          Find_R1_R2_Prio       (void);
1369
```

```
422  *********************************************************************************************************
423  *                                          EVENT CONTROL BLOCK
424  *********************************************************************************************************
425  */
426
427  #if OS_LOWEST_PRIO <= 63u
428  typedef  INT8U    OS_PRIO;
429  #else
430  typedef  INT16U   OS_PRIO;
431  #endif
432
433  #if (OS_EVENT_EN) && (OS_MAX_EVENTS > 0u)
434  typedef struct os_event {
435      INT8U    OSEventType;                /* Type of event control block (see OS_EVENT_TYPE_xxxx)    */
436      void    *OSEventPtr;                 /* Pointer to message or queue structure                   */
437      INT16U   OSEventCnt;                 /* Semaphore Count (not used if other EVENT type)          */
438      OS_PRIO  OSEventGrp;                 /* Group corresponding to tasks waiting for event to occur */
439      OS_PRIO  OSEventTbl[OS_EVENT_TBL_SIZE]; /* List of tasks waiting for event to occur             */
440      //AddedCodePA3part2
441      int OSEventEnterMutexCnt;            /* To know alreaedy enter event or not */
442
443  #if OS_EVENT_NAME_EN > 0u
444      INT8U   *OSEventName;
445  #endif
446  } OS_EVENT;
447  #endif
```

在宣告完變數之後會在 os_task.c 下進行初始化，會先宣告 TaskNumber 去記錄創建的任務數量，方便作優先權的給予，為了避免發生優先權重複的情形發生，因此在創建任務時原本都是以優先權=週期，按照創建的 task 順序依序給予63、62、61、60...等等的優先權。

```
33    #define  MICRIUM_SOURCE
34
35   #ifndef  OS_MASTER_FILE
36    #include <ucos_ii.h>
37    #endif
38    int TaskNumber = 0;      //AddedCodePA3part2
```

```
386         while (ptr != NULL)
387         {
388             TaskInfo[k] = atoi(ptr);
389             ptr = strtok_s(NULL, " ", &pTmp);
390             /*printf("Info: %d\n", task_inf[i]);*/
391             if (k == 0) {
392                 TASK_NUMBER++;
393                 TaskParameter[1].TaskID = TASK_NUMBER;
394             }
395             else if (k == 1) {
396                 TaskParameter[1].TaskArriveTime = TaskInfo[k];
397             }
398             else if (k == 2) {
399                 TaskParameter[1].TaskExecutionTime = TaskInfo[k];
400             }
401             else if (k == 3) {
402                 TaskParameter[1].TaskPeriodic = TaskInfo[k];
403                 TaskParameter[1].TaskPriority = 63 - TaskNumber; //Initial Priority=Period, AddedCodePA3part2
404                 prio = 63 - TaskNumber;
405             }
406             //AddedCodePA3part2
407             else if (k == 4) {
408                 TaskParameter[1].TaskR1Lock = TaskInfo[k];
409             }
410             else if (k == 5) {
411                 TaskParameter[1].TaskR1Unlock = TaskInfo[k];
412             }
413             else if (k == 6) {
414                 TaskParameter[1].TaskR2Lock = TaskInfo[k];
415             }
416             else if (k == 7) {
417                 TaskParameter[1].TaskR2Unlock = TaskInfo[k];
418             }
419             k++;
420         }
421         l++;
422     }
423     fclose(fp);
424     if (prio != 63) {
425         prio = 63 - TaskNumber;
426     }
427     ////////////////////
428     TaskNumber++;
```

之後也需在 os_core.c 的 OS_TCBInit 中去初始化新宣告的參數，像是將 OSTCBPrio 改成等於新賦予的值而並非週期，基本上所做的事和前面 os_task.c 建立 task 時所做的事情類似，同樣的在讀檔時根據創建的任務數去從優先權最低去給予，並初始化 R1、R2 相關的Lock time 與 Unlock time，同時由於讀檔時不包含 idle task 因此優先權最低是從 62 開始給予，而 idle task 是每次執行都最優先建立的任務。

```
2318    OS_ENTER_CRITICAL();
2319    ptcb = OSTCBFreeList;                          /* Get a free TCB from the free TCB list    */
2320    if (ptcb != (OS_TCB *)0) {
2321        OSTCBFreeList          = ptcb->OSTCBNext;  /* Update pointer to free TCB list          */
2322        OS_EXIT_CRITICAL();
2323        ptcb->OSTCBStkPtr      = pstk;             /* Load Stack pointer in TCB                */
2324        ptcb->OSTCBPrio        = prio;             /* Load task priority into TCB              *///AddedCodePA3part2
2325        ptcb->OSTCBOriPrio = prio;                 //AddedCodePA3part1, AddedCodePA3part2
2326        ptcb->OSTCBStat        = OS_STAT_RDY;      /* Task is ready to run                     */
2327        ptcb->OSTCBStatPend    = OS_STAT_PEND_OK;  /* Clear pend status                        */
2328        ptcb->OSTCBDly         = 0u;               /* Task is not delayed                      */
```

```
2359    while (!feof(fp) && prio != 63)
2360    {
2361        k = 0;
2362        memset(str, 0, sizeof(str));
2363        fgets(str, sizeof(str) - 1, fp);
2364        ptr = strtok_s(str, " ", &spTmp);
2365        while (ptr != NULL)
2366        {
2367            TaskInfo[k] = atoi(ptr);
2368            ptr = strtok_s(NULL, " ", &spTmp);
2369            /*printf("Info: %d\n", task_inf[i]);*/
2370            if (k == 0) {
2371                TASK_NUMBER++;
2372                TaskParameter[1].TaskID = TASK_NUMBER;
2373            }
2374            else if (k == 1) {
2375                TaskParameter[1].TaskArriveTime = TaskInfo[k];
2376            }
2377            else if (k == 2) {
2378                TaskParameter[1].TaskExecutionTime = TaskInfo[k];
2379            }
2380            else if (k == 3) {
2381                TaskParameter[1].TaskPeriodic = TaskInfo[k];
2382                TaskParameter[1].TaskPriority = 63-TaskNum; //Initial Priority=Period, AddedCodePA3part2
2383                ptcb->OSTCBPrio = 63 - TaskNum;                    //AddedCodePA3part2
2384                ptcb->OSTCBOriPrio = 63 - TaskNum;                 //AddedCodePA3part1, AddedCodePA3part2
2385                prio = 63 - TaskNum;
```

```
2405    if (prio != 63) {
2406        unsigned int delay = TaskParameter[id - 1].TaskArriveTime;
2407        unsigned int exetime = TaskParameter[id - 1].TaskExecutionTime;
2408        ptcb->OSTCBCyclesExecution = exetime;
2409        ptcb->OSTCBCyclesCount = 0u;
2410        ptcb->OSTCBJobNumber = 0u;
2411        ptcb->OSTCBCyclesEnd = 0u;
2412        ptcb->OSTCBCyclesSwitchStart = 0u;
2413        ptcb->OSTCBMyTaskCtxTimes = 0u;
2414        ptcb->OSTCBCyclesPeriod = TaskParameter[id - 1].TaskPeriodic;
2415        ptcb->OSTCBCyclesArrive = TaskParameter[id - 1].TaskArriveTime;
2416        //AddedCodePA3part2
2417        ptcb->OSTCBR1Lock = TaskParameter[id - 1].TaskR1Lock;
2418        ptcb->OSTCBR1Unlock = TaskParameter[id - 1].TaskR1Unlock;
2419        ptcb->OSTCBR2Lock = TaskParameter[id - 1].TaskR2Lock;
2420        ptcb->OSTCBR2Unlock = TaskParameter[id - 1].TaskR2Unlock;
2421        ptcb->OSTCBUsingResource = 0;
2422        ptcb->OSTCBDeadline = ptcb->OSTCBCyclesArrive + TaskParameter[id - 1].TaskPeriodic;
2423        while (ptcb->OSTCBDly != delay) {
2424            ptcb->OSTCBDly++;
2425        }
2426    }
```

```
2503    //ModifyCodePAlpart2
2504    if (prio == 63 || ptcb->OSTCBCyclesArrive == 0u) {
2505        OSRdyGrp |= ptcb->OSTCBBitY;                /* Make task ready to run                   */
2506        OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
2507    }
2508
2509
2510    OSTaskCtr++;                                    /* Increment the #tasks counter             */
2511    OS_TRACE_TASK_READY(ptcb);
2512    OS_EXIT_CRITICAL();
2513
2514    //AddedCodePA3part1, AddedCodePA3part2
2515    if (prio == 63) {
2516        TotalPrio[TaskNum] = prio;
2517        TotalPeriod[TaskNum] = 0;
2518    }
2519    else {
2520        TotalPeriod[TaskNum] = TaskParameter[id - 1].TaskPeriodic;
2521        TotalPrio[TaskNum] = 63 - TaskNum;
2522    }
```

事前的一些參數宣告即設定都完成之後， 在os_core.c 中撰寫兩個新的副程式去協助任務優先權的排序以及優先權的重新給予，因為一開始在OS_TCBInit給予優先權只考慮避免重複，未按照週期大小給予，因此需要重新初始化。

在副程式 Sort_Prio 中，一開始在 FixedPrio 會先藉由迴圈初始化成放置規定的優先權，也就是 4 的倍數，並再下面那個迴圈進行 sort，每次都把週期最小的任務找出來並依序將優先權透過 OSTaskChangePrio 去做更改。

```
/*//AddedCodePA3part2
*********************************************************************************************************
*                                        Change Priority
*
* Description: Sort the priority base on period, and the priority needs to be 4, 8, 12, 16...
*
* Arguments  : None
*
* Returns    : None
*
* Note       :
*********************************************************************************************************
*/
void Sort_Prio(void) {
    int i;
    int j;
    int save;
    int FindPeriod[10];
    int MinPeriod = 99999;
    int MinIdx;

    for (i = 0; i < 10; i++) {          // Initialize Parameter FixedPrio, FindPrio
        FixedPrio[i] = (i + 1) * 4;
        FindPeriod[i] = TotalPeriod[i];
    }

    for (i = 0; i < TaskNum - 1; i++) {     // Sort the priority base on the period
        MinPeriod = 99999;

        for (j = 0; j < TaskNum; j++) {
            if (FindPeriod[j] < MinPeriod && MinPeriod != 0) {
                MinPeriod = FindPeriod[j];
                MinIdx = j;
            }
            else if (MinPeriod == 0){
                MinPeriod = FindPeriod[j];
                MinIdx = j;
            }
        }
        //SortedPeriod[MinIdx] = MinPeriod;
        if (MinPeriod != 0) {
            OSTaskChangePrio(TotalPrio[MinIdx], FixedPrio[i]); // OSTaskChangePrio(Origin, Want)
            TotalPrio[MinIdx] = FixedPrio[i];
        }
        FindPeriod[MinIdx] = 99999;
    }
}
```

在副程式 Find_R1_R2_PRIO 中主要負責初始化 R1 和 R2 的優先權，因為他們的優先權必須要由有用到資源的最高任務去決定，因此在這個副程式中會去找出使用到該資源的最高優先全為何，並判斷是否兩個資源都被同一個優先權最高的任務使用，若為是則保持 R1 優先權要較高故等於 task 優先權 -3 ，R2在這時則會等於該任務的優先權 -1。

```
void Find_R1_R2_Prio(void) {
    int i;
    R1_PRIO = 62;
    R2_PRIO = 62;
    OS_TCB* ptcb;

    Sort_Prio();

    for (i = 1; i < TaskNum; i++) {
        ptcb = OSTCBPrioTbl[TotalPrio[i]];

        if (ptcb->OSTCBR1Lock != 0) {
            if (TotalPrio[i] < R1_PRIO)
                R1_PRIO = TotalPrio[i];
        }

        if (ptcb->OSTCBR2Lock != 0) {
            if (TotalPrio[i] < R2_PRIO)
                R2_PRIO = TotalPrio[i];
        }
    }
    if (R1_PRIO == R2_PRIO) {
        R1_PRIO = R1_PRIO - 3;
        R2_PRIO = R2_PRIO - 1;
    }
    else {
        R1_PRIO = R1_PRIO - 1;
        R2_PRIO = R2_PRIO - 1;
    }
}
```

最後則是修改 os_mutex.c 中的 OSMutexPend 以及 OSMutexPost：

在 OSMutexCreate 中，為了記錄是否為第一次進入 Mutex 因此要在建立Event時將 OSEventEnterMutexCnt 先初始化成 0。

```
246         OS_EXIT_CRITICAL();
247         pevent->OSEventType = OS_EVENT_TYPE_MUTEX;
248         pevent->OSEventCnt  = (INT16U)((INT16U)prio << 8u) | OS_MUTEX_AVAILABLE; /* Resource is avail.     */
249         pevent->OSEventPtr  = (void *)0;                    /* No task owning the mutex               */
250         pevent->OSEventEnterMutexCnt = 0;                   //AddedCodePA3part2
```

而在 OSMutexPend 中，會先新增一判斷，當 OSEventEnterMutexCnt =1時代表並非第一次進入 OSMutexPend 中，會直接進行 return ，並新增打印相關的程式碼，使其打印符合格式，除此之外由於 OSMutexPend 原本只會在有 task 要進去搶資源時才會將搶到資源的 task 優先權提升，因此要將優先權提升的相關程式碼更改位置上移至進來時就將搶到資源的 task 的優先權先提升。

```
470 void  OSMutexPend (OS_EVENT  *pevent,
471                    INT32U     timeout,
472                    INT8U     *perr)
473 {
474     INT8U      pcp;                                     /* Priority Ceiling Priority (PCP)          */
475     INT8U      mprio;                                   /* Mutex owner priority                     */
476     BOOLEAN    rdy;                                     /* Flag indicating task was ready           */
477     OS_TCB    *ptcb;
478     OS_EVENT  *pevent2;
479     INT8U      y;
480 #if OS_CRITICAL_METHOD == 3u                            /* Allocate storage for CPU status register */
481     OS_CPU_SR  cpu_sr = 0u;
482 #endif
483
484
485 #ifdef OS_SAFETY_CRITICAL
486     if (perr == (INT8U *)0) {
487         OS_SAFETY_CRITICAL_EXCEPTION();
488         return;
489     }
490 #endif
491
492
493     if (pevent->OSEventEnterMutexCnt == 1)  //AddedCodePA3part2
494         return;
495
496 #if OS_ARG_CHK_EN > 0u
497     if (pevent == (OS_EVENT *)0) {                      /* Validate 'pevent'                        */
498         *perr = OS_ERR_PEVENT_NULL;
499         return;
500     }
501 #endif
502
503     OS_TRACE_MUTEX_PEND_ENTER(pevent, timeout);
504
505     if (pevent->OSEventType != OS_EVENT_TYPE_MUTEX) {   /* Validate event block type                */
506         *perr = OS_ERR_EVENT_TYPE;
507         OS_TRACE_MUTEX_PEND_EXIT(*perr);
508         return;
509     }
510     if (OSIntNesting > 0u) {                            /* See if called from ISR ...               */
511         *perr = OS_ERR_PEND_ISR;                        /* ... can't PEND from an ISR               */
512         OS_TRACE_MUTEX_PEND_EXIT(*perr);
513         return;
514     }
515     if (OSLockNesting > 0u) {                           /* See if called with scheduler locked ...  */
516         *perr = OS_ERR_PEND_LOCKED;                     /* ... can't PEND when locked               */
517         OS_TRACE_MUTEX_PEND_EXIT(*perr);
518         return;
519     }
520
521     pcp = (INT8U)(pevent->OSEventCnt >> 8u);            /* Get PCP from mutex                       */
522
523     OS_ENTER_CRITICAL();
524
525
526     //AddedCodePA3part2
527     if (pevent->OSEventEnterMutexCnt == 0) {
528         if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
529             if ((OSTimeGet() == (OSTCBCur->OSTCBCyclesStart + OSTCBCur->OSTCBR2Lock - 1)) && OSTCBCur->OSTCBR2Lock != 0) {
530                 printf("%2d\ttask%d get R2 \t\t\t\t\t\t", OSTimeGet(), OSTCBCur->OSTCBId);
531                 fprintf(Output_fp, "%2d\ttask%d get R2 \t\t\t\t\t\t", OSTimeGet(), OSTCBCur->OSTCBId);
532                 OSTCBCur->OSTCBUsingResource++;
533             }
534             else if ((OSTimeGet() == (OSTCBCur->OSTCBCyclesStart + OSTCBCur->OSTCBR1Lock - 1)) && OSTCBCur->OSTCBR1Lock != 0) {
535                 printf("%2d\ttask%d get R1 \t\t\t\t\t\t", OSTimeGet(), OSTCBCur->OSTCBId);
536                 fprintf(Output_fp, "%2d\ttask%d get R1 \t\t\t\t\t\t", OSTimeGet(), OSTCBCur->OSTCBId);
537                 OSTCBCur->OSTCBUsingResource++;
538             }
539         }fclose(Output_fp);
540         pevent->OSEventEnterMutexCnt++;
541     }
```

```
542                                             /* Is Mutex available?                      */
543     if ((INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8) == OS_MUTEX_AVAILABLE) {
544         pevent->OSEventCnt &= OS_MUTEX_KEEP_UPPER_8;        /* Yes, Acquire the resource            */
545         pevent->OSEventCnt |= OSTCBCur->OSTCBPrio;          /*      Save priority of owning task    */
546         pevent->OSEventPtr  = (void *)OSTCBCur;             /*      Point to owning task's OS_TCB   */
547
548         //Here
549         mprio = (INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8); /*  Get priority of mutex owner   */
550         ptcb = (OS_TCB*)(pevent->OSEventPtr);                 /*    Point to TCB of mutex owner   */
551
552         if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
553             if ((OSTimeGet() == (OSTCBCur->OSTCBCyclesStart + OSTCBCur->OSTCBR2Lock - 1)) && OSTCBCur->OSTCBR2Lock != 0) {
554                 if (ptcb->OSTCBPrio > pcp) {
555                     printf(" %2d to %d\n", OSTCBCur->OSTCBPrio, pcp);
556                     fprintf(Output_fp, " %2d to %d\n", OSTCBCur->OSTCBPrio, pcp);
557                 }
558                 else {
559                     printf(" %2d to %d\n", OSTCBCur->OSTCBPrio, OSTCBCur->OSTCBPrio);
560                     fprintf(Output_fp, " %2d to %d\n", OSTCBCur->OSTCBPrio, OSTCBCur->OSTCBPrio);
561                 }
562             }
563             else if ((OSTimeGet() == (OSTCBCur->OSTCBCyclesStart + OSTCBCur->OSTCBR1Lock - 1)) && OSTCBCur->OSTCBR1Lock != 0) {
564                 if (ptcb->OSTCBPrio > pcp) {
565                     printf(" %2d to %d\n", OSTCBCur->OSTCBPrio, pcp);
566                     fprintf(Output_fp, " %2d to %d\n", OSTCBCur->OSTCBPrio, pcp);
567                 }
568                 else {
569                     printf(" %2d to %d\n", OSTCBCur->OSTCBPrio, OSTCBCur->OSTCBPrio);
570                     fprintf(Output_fp, " %2d to %d\n", OSTCBCur->OSTCBPrio, OSTCBCur->OSTCBPrio);
571                 }
572             }
573         }fclose(Output_fp);
```

同意的在 OSMutexPost 也為了避免瘋狂打印會去判斷若 OSEventEnterMutexCnt =0 時代表並非第一次進入這個副程式，就直接 return 跳出，至於其餘有新增的程式碼則是與打印相關的程式碼。

```
750 INT8U  OSMutexPost (OS_EVENT *pevent)
751 {
752     INT8U      pcp;                                 /* Priority ceiling priority                */
753     INT8U      prio;
754 #if OS_CRITICAL_METHOD == 3u                        /* Allocate storage for CPU status register     */
755     OS_CPU_SR  cpu_sr = 0u;
756 #endif
757
758
759     if (pevent->OSEventEnterMutexCnt == 0)  //AddedCodePA3part2
760         return;
761
762     if (OSIntNesting > 0u) {                         /* See if called from ISR ...               */
763         return (OS_ERR_POST_ISR);                    /* ... can't POST mutex from an ISR         */
764     }
765 #if OS_ARG_CHK_EN > 0u
766     if (pevent == (OS_EVENT *)0) {                   /* Validate 'pevent'                        */
767         return (OS_ERR_PEVENT_NULL);
768     }
769 #endif
770
771     OS_TRACE_MUTEX_POST_ENTER(pevent);
772
773     if (pevent->OSEventType != OS_EVENT_TYPE_MUTEX) { /* Validate event block type               */
774         OS_TRACE_MUTEX_POST_EXIT(OS_ERR_EVENT_TYPE);
775         return (OS_ERR_EVENT_TYPE);
776     }
777     OS_ENTER_CRITICAL();
778
779     //AddedCodePA3part2
780     if (pevent->OSEventEnterMutexCnt == 1) {
781         if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
782             if ((OSTCBCur->OSTCBCyclesCount + 1 == OSTCBCur->OSTCBR1Unlock) && OSTCBCur->OSTCBR1Unlock != 0) {
783                 printf("%2d\ttask%d release R1  \t\t\t\t", OSTimeGet() + 1, OSTCBCur->OSTCBId);
784                 fprintf(Output_fp, "%2d\ttask%d release R1  \t\t\t\t", OSTimeGet() + 1, OSTCBCur->OSTCBId);
785                 OSTCBCur->OSTCBUsingResource--;
786             }
787             else if ((OSTCBCur->OSTCBCyclesCount + 1 == OSTCBCur->OSTCBR2Unlock) && OSTCBCur->OSTCBR2Unlock != 0) {
788                 printf("%2d\ttask%d release R2  \t\t\t\t", OSTimeGet() + 1, OSTCBCur->OSTCBId);
789                 fprintf(Output_fp, "%2d\ttask%d release R2  \t\t\t\t", OSTimeGet() + 1, OSTCBCur->OSTCBId);
790                 OSTCBCur->OSTCBUsingResource--;
791             }fclose(Output_fp);
792         }
793         pevent->OSEventEnterMutexCnt--;
794     }
795
796     pcp  = (INT8U)(pevent->OSEventCnt >> 8u);        /* Get priority ceiling priority of mutex   */
797     prio = (INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8);  /* Get owner's original priority    */
798     if (OSTCBCur != (OS_TCB *)pevent->OSEventPtr) {   /* See if posting task owns the MUTEX          */
799         OS_EXIT_CRITICAL();
800         OS_TRACE_MUTEX_POST_EXIT(OS_ERR_NOT_MUTEX_OWNER);
801         return (OS_ERR_NOT_MUTEX_OWNER);
802     }
803
804     if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
805         if (OSTCBCur->OSTCBPrio == pcp) {
806             printf(" %2d to %d\n", OSTCBCur->OSTCBPrio, prio);
807             fprintf(Output_fp, " %2d to %d\n", OSTCBCur->OSTCBPrio, prio);
808         }
809         else {
810             printf(" %2d to %d\n", OSTCBCur->OSTCBPrio, prio);
811             fprintf(Output_fp, " %2d to %d\n", OSTCBCur->OSTCBPrio, prio);
812         }
813     }fclose(Output_fp);
```

最後則是針對 main program 的撰寫，要注意的是在創建 R1、R2 的 OSMutexCreate 要放置在 task 創建完成之後，因為此時 R1、R2 的 priority 也初始化完成。

```
114     for (n = 0; n < TASK_NUMBER; n++) {
115         Task_STK[n] = malloc(TASK_STACKSIZE * sizeof(int));
116         OSTaskCreateExt(task,                       /*Create the task2*/
117             &TaskParameter[n],
118             &Task_STK[n][TASK_STACKSIZE - 1],
119             TaskParameter[n].TaskPriority,
120             TaskParameter[n].TaskID,
121             &Task_STK[n][0],
122             TASK_STACKSIZE,
123             &TaskParameter[n],
124             (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));
125     }
126
127     //AddedCodePA3part1
128     INT8U err;
129     //R1 = OSSemCreate(1);
130     //R2 = OSSemCreate(1);
131
132     //AddedCodePA3part2
133     Find_R1_R2_Prio();
134     R1 = OSMutexCreate(R1_PRIO, &err);
135     R2 = OSMutexCreate(R2_PRIO, &err);
136
```

在 task 執行的 while 迴圈中，因為會出現 task 搶到2個資源的情形，因此在 semaphore 所撰寫的資源 release 的判斷式在 Mutex 不適用，需改成當該任務已執行的秒數=資源 unlock 的時間時才能釋放資源。

```
239 //AddedCodePA3part2
240 void task(void* pdata) {
241     INT8U err;
242     while (1) {
243
244         if ((OSTimeGet() == (OSTCBCur->OSTCBCyclesStart + OSTCBCur->OSTCBR2Lock - 1)) && OSTCBCur->OSTCBR2Lock != 0) {
245             OSMutexPend(R2, 0, &err);
246         }
247         else if ((OSTimeGet() == (OSTCBCur->OSTCBCyclesStart + OSTCBCur->OSTCBR1Lock - 1)) && OSTCBCur->OSTCBR1Lock != 0) {
248             OSMutexPend(R1, 0, &err);
249         }
250         else if ((OSTCBCur->OSTCBCyclesCount + 1 == OSTCBCur->OSTCBR1Unlock) && OSTCBCur->OSTCBR1Unlock != 0) {
251             OSMutexPost(R1);
252         }
253         else if ((OSTCBCur->OSTCBCyclesCount + 1 == OSTCBCur->OSTCBR2Unlock) && OSTCBCur->OSTCBR2Unlock != 0) {
254             OSMutexPost(R2);
255         }
256
257         if (OSTCBCur->OSTCBCyclesCount == OSTCBCur->OSTCBCyclesExecution) {
258             OSTCBCur->OSTCBCyclesCount = 0;
259             OSTimeDly(OSTCBCur->OSTCBDly);
260         }
261     }
262 }
263
```

[PART III] Performance Analysis [10%]

Compare the scheduling behaviors between NPCS and CPP with PART I and PART II results.

NPCS 因為在搶到資源後就會禁止在執行資源時被其他 task 打斷，雖然可以完整執行完資源，但也因此可能導致 high priority task 在 ready 之後因為 low priority task 正在使用資源而無法遲遲執行，似 Example 1 的狀況，此時任務最多都只有使用到一個資源，而 CPP 在使用 R1 資源時是可以被打斷去執行 high priority task (Example 1 中 的 T2)。

而由 Example 2 去做討論，CPP 會賦予 R1、R2 資源優先權 ，並該優先權是由使用到該資源且優先權最高的 task 去決定，因此若是 high priority task 使用到許多資源，那麼會盡快使 low priority task 將拿到的資源用完釋放，避免拉長 priority task 的完成時間。

Explain how NPCS and CPP avoid the deadlock problem.

NPCS 會使搶到資源的 task 優先執行並釋放，像是拿到 R1 資源就要等 R1 資源用完才有可能才會換其他 task 執行，因此若是在使用 R1 過程中穿插使用了其他資源也不會因此打斷該 task 的執行，所以能避免 deadlock 的發生。

在 CPP 中，會賦予資源優先權，並在 task 搶到資源時提升 task 的 priority ，且優先權是由使用到該資源且優先權最高的 task 去決定，可以讓搶到資源的 task 在尚未使用完該資源時都保持較高的優先權，因此能免 task 雙方拿著未使用完的資源，卻需要他人正在執行的資源狀況 (deadlock) 的發生。

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**).