RTOS-PA3

學號：M11007326 姓名：黃鈞臨

[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%)

Example Task Set 1 = { Task1 (1, 1, 8, 60, 1, 6, 0, 0),
Task2 (2, 8, 5, 30, 0, 0, 1, 3),
Task3 (3, 0, 6, 20, 0, 0, 0, 0) }

```
 6      Completion      task( 3)( 0)   task( 1)( 0)
 7       task 1 get R1
12       task 1 release R1
12      Preemption      task( 1)( 0)   task( 2)( 0)
13       task 2 get R2
15       task 2 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       task 2 get R2
41       task 2 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       task 1 get R1
72       task 1 release R1
72      Preemption      task( 1)( 1)   task( 2)( 2)
73       task 2 get R2
75       task 2 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       task 2 get R2
```

Example Task Set 2 = { Task1 (1, 2, 8, 20, 2, 7, 4, 6),
Task2 (2, 0, 11, 40, 5, 8, 1, 9)}

```
OSTick    created, Thread ID  1988
Tick   Event         CurrentTask ID   NextTask ID
 1      task 2 get R2
 5      task 2 get R1
 8      task 2 release R1
 9      task 2 release R2
 9     Preemption      task( 2)( 0)   task( 1)( 0)
11      task 1 get R1
13      task 1 get R2
15      task 1 release R2
16      task 1 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      task 1 get R1
26      task 1 get R2
28      task 1 release R2
29      task 1 release R1
30     Completion      task( 1)( 1)   task(63)
40     Preemption      task(63)       task( 2)( 1)
41      task 2 get R2
45      task 2 get R1
48      task 2 release R1
49      task 2 release R2
49     Preemption      task( 2)( 1)   task( 1)( 2)
51      task 1 get R1
53      task 1 get R2
55      task 1 release R2
56      task 1 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      task 1 get R1
66      task 1 get R2
```

```
68      task 1 release R2
69      task 1 release R1
70     Completion      task( 1)( 3)   task(63)
80     Preemption      task(63)       task( 2)( 2)
81      task 2 get R2
85      task 2 get R1
88      task 2 release R1
89      task 2 release R2
89     Preemption      task( 2)( 2)   task( 1)( 4)
91      task 1 get R1
93      task 1 get R2
95      task 1 release R2
96      task 1 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). (20%)

```
#define INFO 10 //information of task
#define R1_PRIO 3
#define R2_PRIO 4


    /*Input File*/

    /*Output file*/
    FILE* Output_fp;
    errno_t Output_err;
    /*Output file*/

    /*Task structure*/
    typedef struct task_para_set {
        INT16U TaskID;
        INT16U TaskArriveTime;
        INT16U TaskExecutionTime;
        INT16U TaskPeriodic;
        INT16U TaskNumber;
        INT16U TaskPriority;
        INT16U R1lock;
        INT16U R1unlock;
        INT16U R2lock;
        INT16U R2unlock;
        //INT16U TaskJobNum;
    } task_para_set;
```

在 ucos_ii.h 裡面，設定INFO的大小，並在Task_para_set裡面興建R1lock等變數，以便紀錄資訊。

```
for (int p = 1; p < TASK_NUMBER; p++) {
    task_para_set key = TaskParameter[p];
    int sp = p - 1;
    while (key.TaskPeriodic < TaskParameter[sp].TaskPeriodic && sp >= 0) {
        TaskParameter[sp + 1] = TaskParameter[sp];
        sp--;
    }
    TaskParameter[sp + 1] = key;
}
for (int p = 0; p < TASK_NUMBER; p++) {
    TaskParameter[p].TaskPriority = p;
}
```

在app_hooks裡面，由於RM排程裡週期短的優先權高，因此先利用insertion sort

排序週期後在根據位置給予相對應的優先權。

```
OS_EVENT* R1;
OS_EVENT* R2;

R1 = OSSemCreate(1);
R2 = OSSemCreate(1);
for (n = 0; n < TASK_NUMBER; n++) {
    OSTaskCreateExt(task1,
        &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),
        TaskParameter[n].TaskExecutionTime,
        TaskParameter[n].TaskArriveTime,
        TaskParameter[n].R1lock,
        TaskParameter[n].R1unlock,
        TaskParameter[n].R2lock,
        TaskParameter[n].R2unlock,
        TaskParameter[n].TaskPeriodic);
}
```

在main.c裡面，需要先設定一些內容，例如建立OS_EVENT，OSSemCreate()裡面要設定sem的cnt，若cnt > 0的話sem才能夠pend，因此這邊都先設定為1，最後，為了方便，我把R1lock等資訊也都放入了TCB裡面。

```
if (OSTCBCur->OSTCBR1UnLock != 0 || OSTCBCur->OSTCBR2UnLock != 0) {

    if (OSTCBCur->OSTCBWorkTime == OSTCBCur->OSTCBR1Lock && r1flag == 0 && OSTCBCur->OSTCBR1UnLock!=0){
        printf("%2d\ttask %d get R1\n", OSTimeGet(), OSTCBCur->OSTCBId);
        if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0){ ... }
        OSSchedLock();
        OSSemPend(R1, 0, &err);

        r1flag = 1;
    }
    if (OSTCBCur->OSTCBWorkTime == OSTCBCur->OSTCBR1UnLock && r1flag == 1) {
        printf("%2d\ttask %d release R1\n", OSTimeGet(), OSTCBCur->OSTCBId);
        if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0){ ... }
        OSSemPost(R1);
        OSSchedUnlock();
        r1flag = 0;
    }
    if (OSTCBCur->OSTCBWorkTime == OSTCBCur->OSTCBR2Lock && r2flag == 0 && OSTCBCur->OSTCBR2UnLock != 0)
        printf("%2d\ttask %d get R2\n", OSTimeGet(), OSTCBCur->OSTCBId);
        if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0){ ... }
        OSSchedLock();
        OSSemPend(R2, 0, &err);
        r2flag = 1;
    }
    if (OSTCBCur->OSTCBWorkTime == OSTCBCur->OSTCBR2UnLock && r2flag == 1) {
        printf("%2d\ttask %d release R2\n", OSTimeGet(), OSTCBCur->OSTCBId);
        if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0){ ... }
        OSSemPost(R2);
        OSSchedUnlock();
        r2flag = 0;
    }

}
```

最後，是 Task 內的程式，藍色框框是為了判斷當前Task有沒有使用到semaphore，如果有使用到的話，R1Unlock或R2Unlock一定會有一個不是零。

綠色框框是為了判斷當下時間有沒有要使用到semaphore，worktime代表的是Task使用cpu的時間，若等於r1lock的話就代表要使用r1這個資源，後面的r1flag是為了防止同一個Task重複索取semaphore，預設為0，若拿到資源則會被改為1，釋放資源又會回到0。

紅色框框是執行拿取資源跟釋放資源的地方，而NPCS要在資源使用的時候避免被其他任務搶占，因此我在pend前使用了OSSchedlock()、post前使用了OSSchedUnlock()，透過關閉排程的方式實做NPCS。

[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%)

Example Task Set 1 = { Task1 (1, 1, 8, 60, 1, 6, 0, 0),

Task2 (2, 8, 5, 30, 0, 0, 1, 3),

Task3 (3, 0, 6, 20, 0, 0, 0, 0) }

```
Tick   Event         CurrentTask ID   NextTask ID
 6     Completion      task( 3)( 0)   task( 1)( 0)
 7      task 1 get R1                                  12 to 11
 8     Preemption      task( 1)( 0)   task( 2)( 0)
 9      task 2 get R2                                   8 to  7
11      task 2 release R2                                      7 to  8
13     Completion      task( 2)( 0)   task( 1)( 0)
17      task 1 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      task 2 get R2                                   8 to  7
40     Preemption      task( 2)( 1)   task( 3)( 2)
46     Completion      task( 3)( 2)   task( 2)( 1)
47      task 2 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      task 1 get R1                                  12 to 11
68     Preemption      task( 1)( 1)   task( 2)( 2)
69      task 2 get R2                                   8 to  7
71      task 2 release R2                                      7 to  8
73     Completion      task( 2)( 2)   task( 1)( 1)
77      task 1 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      task 2 get R2                                   8 to  7
100     Preemption      task( 2)( 3)   task( 3)( 5)
```

Example Task Set 2 = { Task1 (1, 2, 8, 20, 2, 7, 4, 6),

Task2 (2, 0, 11, 40, 5, 8, 1, 9)}

Tick	Event	CurrentTask ID	NextTask ID				
1	task 2 get R2			8	to	3	
5	task 2 get R1			3	to	1	
8	task 2 release R1					1	to 3
9	task 2 release R2					3	to 8
9	Preemption	task(2)(0)	task(1)(0)				
11	task 1 get R1			4	to	1	
13	task 1 get R2			1	to	1	
15	task 1 release R2					1	to 1
16	task 1 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	task 1 get R1			4	to	1	
26	task 1 get R2			1	to	1	
28	task 1 release R2					1	to 1
29	task 1 release R1					1	to 4
30	Completion	task(1)(1)	task(63)				
40	Preemption	task(63)	task(2)(1)				
41	task 2 get R2			8	to	3	
45	task 2 get R1			3	to	1	
48	task 2 release R1					1	to 3
49	task 2 release R2					3	to 8
49	Preemption	task(2)(1)	task(1)(2)				
51	task 1 get R1			4	to	1	
53	task 1 get R2			1	to	1	
55	task 1 release R2					1	to 1
56	task 1 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	task 1 get R1			4	to	1	
66	task 1 get R2			1	to	1	
68	task 1 release R2					1	to 1
69	task 1 release R1					1	to 4
70	Completion	task(1)(3)	task(63)				
80	Preemption	task(63)	task(2)(2)				
81	task 2 get R2			8	to	3	
85	task 2 get R1			3	to	1	
88	task 2 release R1					1	to 3
89	task 2 release R2					3	to 8
89	Preemption	task(2)(2)	task(1)(4)				
91	task 1 get R1			4	to	1	
93	task 1 get R2			1	to	1	
95	task 1 release R2					1	to 1
96	task 1 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). (30%)

```
for (int p = 0; p < TASK_NUMBER; p++) {
    TaskParameter[p].TaskPriority = p * 4 + 4;
}
/*read file*/
int r1max = 63;
for (int p = 0; p < TASK_NUMBER; p++) {
    if (TaskParameter[p].R1unlock != 0) {
        if (r1max > TaskParameter[p].TaskPriority) {
            r1max = TaskParameter[p].TaskPriority-1;
        }
    }
}

int r2max = 63;
for (int p = 0; p < TASK_NUMBER; p++) {
    if (TaskParameter[p].R2unlock != 0) {
        if (r2max > TaskParameter[p].TaskPriority) {
            r2max = TaskParameter[p].TaskPriority-1;
        }
    }
}
if (r2max == r1max)
    r1max = r1max - 2;
R1MAXPrio = r1max;
R2MAXPrio = r2max;
```

CPP讀檔的部分與NPCS雷同，而有差異的地方在於優先權的分配，因為CPP所使用的semaphore需要獨立的優先權，在這個程式裡，Task的優先權為4的倍數，且最大優先權為4(紅框部分)，接下來的綠框部分為分配semaphore優先權，for迴圈先查詢Task中有用到R1且優先權最大的，並將R1的優先權設為該Task的優先權-1，R2也是如此，最後藍框部分是當R1的優先權等於R2的時候，將R1的優先權在-2以避免重複的情況。

```
OS_EVENT  *OSMutexCreate (INT8U   prio,
                          INT8U  *perr,
                          INT8U  name
    )
{
    OS_EVENT  *pevent;
#if OS_CRITICAL_METHOD == 3u 使用中的前置處理器區塊
#endif



#ifdef OS_SAFETY_CRITICAL 非使用中的前置處理器區塊
#endif

#ifdef OS_SAFETY_CRITICAL_IEC61508 非使用中的前置處理器區塊
#endif

#if OS_ARG_CHK_EN > 0u 使用中的前置處理器區塊
#endif
    if (OSIntNesting > 0u) { ... }
    OS_ENTER_CRITICAL();
    if (prio != OS_PRIO_MUTEX_CEIL_DIS) { ... }

    pevent = OSEventFreeList;                                /* Get next free event control block        */
    if (pevent == (OS_EVENT *)0) { ... }
    OSEventFreeList     = (OS_EVENT *)OSEventFreeList->OSEventPtr; /* Adjust the free list                */
    OS_EXIT_CRITICAL();
    pevent->OSEventType = OS_EVENT_TYPE_MUTEX;
    pevent->OSEventCnt  = (INT16U)((INT16U)prio << 8u) | OS_MUTEX_AVAILABLE; /* Resource is avail.     */
    pevent->OSEventPtr  = (void *)0;                         /* No task owning the mutex                 */
#if OS_EVENT_NAME_EN > 0u
    pevent->OSEventName = (INT8U *)(void *)name;
#endif
    OS_EventWaitListInit(pevent);
    OS_TRACE_MUTEX_CREATE(pevent, pevent->OSEventName);
   *perr = OS_ERR_NONE;
    return (pevent);
}
```

在OSMutexCreate裡面，新增了一個變數name，最後面的EventName也要從問號改成name，目的是為了能夠在pend和post分辨出R1跟R2，方便printf訊息。

```
if ((INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8) == OS_MUTEX_AVAILABLE) {
    pevent->OSEventCnt &= OS_MUTEX_KEEP_UPPER_8;       /* Yes, Acquire the resource                */
    pevent->OSEventCnt |= OSTCBCur->OSTCBPrio;         /*      Save priority of owning task        */
    pevent->OSEventPtr  = (void *)OSTCBCur;            /*      Point to owning task's OS_TCB       */
    if ((pcp != OS_PRIO_MUTEX_CEIL_DIS) &&
        (OSTCBCur->OSTCBPrio <= pcp)) {                /*      PCP 'must' have a SMALLER prio ...  */
         OS_EXIT_CRITICAL();                           /*      ... than current task!              */
        *perr = OS_ERR_PCP_LOWER;
    } else {
         OS_EXIT_CRITICAL();
        *perr = OS_ERR_NONE;
    }
    if (pcp != OS_PRIO_MUTEX_CEIL_DIS) {
        mprio = (INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8); /*  Get priority of mutex owner   */
        ptcb  = (OS_TCB *)(pevent->OSEventPtr);                      /*      Point to TCB of mutex owner   */

        if(pevent->OSEventName == 1){
            printf("%2d\ttask %d get R1", OSTimeGet(), OSTCBCur->OSTCBId);
            if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
            {
                fprintf(Output_fp, "%2d\ttask %d get R1", OSTimeGet(), OSTCBCur->OSTCBId);
                fclose(Output_fp);
            }
        }
        else {
            printf("%2d\ttask %d get R2", OSTimeGet(), OSTCBCur->OSTCBId);
            if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
            {
                fprintf(Output_fp, "%2d\ttask %d get R2", OSTimeGet(), OSTCBCur->OSTCBId);
                fclose(Output_fp);
            }
        }
    }
```

在OSMutexPend裡面，這個if迴圈是當Mutex avaliable的時候會進來的，而他原本預設的寫法是PCP，與CPP的差異在於CPP的ceiling若大於當前的Task的優先權的話，將會把當前的Task的優先權拉高到ceiling，而PCP則是要等到有其他人也想使用這個資源的時候才有可能去拉高優先權，因此我們把原本PCP拉高優先權的部分複製並貼到這個if迴圈裡面，讓資源被拿取的時候就去判斷要不要拉高當前Task的優先權，紅框部分可以看到利用前面MutexCreate加上得name變數，可以幫助我們分辨目前的資源是R1還是R2。

```
if (ptcb->OSTCBPrio > pcp) {                                /*     Need to promote prio of owner?*
    y = ptcb->OSTCBY;
    if ((OSRdyTbl[y] & ptcb->OSTCBBitX) != 0u) {            /*     See if mutex owner is ready   */
        OSRdyTbl[y] &= (OS_PRIO)~ptcb->OSTCBBitX;           /*     Yes, Remove owner from Rdy ...*/
        if (OSRdyTbl[y] == 0u) {                            /*          ... list at current prio */
            OSRdyGrp &= (OS_PRIO)~ptcb->OSTCBBitY;
        }
        rdy = OS_TRUE;
    } else {
        pevent2 = ptcb->OSTCBEventPtr;
        if (pevent2 != (OS_EVENT *)0) {                     /* Remove from event wait list       */
            y = ptcb->OSTCBY;
            pevent2->OSEventTbl[y] &= (OS_PRIO)~ptcb->OSTCBBitX;
            if (pevent2->OSEventTbl[y] == 0u) {
                pevent2->OSEventGrp &= (OS_PRIO)~ptcb->OSTCBBitY;
            }
        }
        rdy = OS_FALSE;                                     /* No                                */
    }
    printf("                    %2d to %2d\n", ptcb->OSTCBPrio, pcp);
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
    {
        fprintf(Output_fp, "                     %2d to %2d\n", ptcb->OSTCBPrio, pcp);
        fclose(Output_fp);
    }
    ptcb->OSTCBPrio = pcp;                                  /* Change owner task prio to PCP     */

    OS_TRACE_MUTEX_TASK_PRIO_INHERIT(ptcb, pcp);
```

接續上面，若Mutex的優先權較高，將會更改任務的優先權，要注意的是，原本PCP排程除了Mutex優先權大於任務的以外，還需要當前Mutex所有者的優先權大於當前任務的優先權才會做交換，但CPP只要符合第一個條件就應該要交換，因此第二個條件我就直接刪掉了，最後，在這個迴圈裡面讓他 printf 當前優先權 to Mutex的優先權。

```
else {
    printf("                    %2d to %2d\n", ptcb->OSTCBPrio, ptcb->OSTCBPrio);
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
    {
        fprintf(Output_fp, "                     %2d to %2d\n", ptcb->OSTCBPrio, ptcb->OSTCBPrio)
        fclose(Output_fp);
    }
```

如果當前的任務優先權大於Mutex的話，就不會更改任務優先權，因此這裡printf 當前優先權to當前優先權。

```
if (pcp != OS_PRIO_MUTEX_CEIL_DIS) {
    mprio = (INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8); /* Get priority of mutex owner   */
    ptcb  = (OS_TCB *)(pevent->OSEventPtr);                      /*     Point to TCB of mutex owner   */

    if (pevent->OSEventName == 1) {
        printf("%2d\ttask %d get R1", OSTimeGet(), OSTCBCur->OSTCBId);
        if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
        {
            fprintf(Output_fp, "%2d\ttask %d get R1", OSTimeGet(), OSTCBCur->OSTCBId);
            fclose(Output_fp);
        }
    }
    else {
        printf("%2d\ttask %d get R2", OSTimeGet(), OSTCBCur->OSTCBId);
        if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
        {
            fprintf(Output_fp, "%2d\ttask %d get R2", OSTimeGet(), OSTCBCur->OSTCBId);
            fclose(Output_fp);
        }
    }


    if (ptcb->OSTCBPrio > pcp) {                                 /*     Need to promote prio of owner?*/
        y = ptcb->OSTCBY;
        if ((OSRdyTbl[y] & ptcb->OSTCBBitX) != 0u) { ... } else {
            pevent2 = ptcb->OSTCBEventPtr;
            if (pevent2 != (OS_EVENT *)0) { ... }
            rdy = OS_FALSE;                                      /* No                            */
        }

        printf("                    %2d to %2d\n", ptcb->OSTCBPrio, pcp);
        if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
        {
            fprintf(Output_fp, "                    %2d to %2d\n", ptcb->OSTCBPrio, pcp);
            fclose(Output_fp);
        }

        ptcb->OSTCBPrio = pcp;                                   /* Change owner task prio to PCP */

        OS_TRACE_MUTEX_TASK_PRIO_INHERIT(ptcb, pcp);
```

若Mutex not available 的話，就會直接考慮要不要交換任務優先權，剛剛有修改過的地方這邊也都要做一樣的修改，OSMutexPend的修改就到這邊結束。

```
if (pevent->OSEventName == 1) {
    printf("%2d\ttask %d release R1", OSTimeGet(), OSTCBCur->OSTCBId);
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
    {
        fprintf(Output_fp, "%2d\ttask %d release R1", OSTimeGet(), OSTCBCur->OSTCBId);
        fclose(Output_fp);
    }
}
else {
    printf("%2d\ttask %d release R2", OSTimeGet(), OSTCBCur->OSTCBId);
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
    {
        fprintf(Output_fp, "%2d\ttask %d release R2", OSTimeGet(), OSTCBCur->OSTCBId);
        fclose(Output_fp);
    }
}
if(pcp<=prio){
    printf("                    %2d to %2d\n", pcp, prio);
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
    {
        fprintf(Output_fp, "                    %2d to %2d\n", pcp, prio);
        fclose(Output_fp);
    }
}
else {
    printf("                    %2d to %2d\n", prio, prio);
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
    {
        fprintf(Output_fp, "                    %2d to %2d\n", prio, prio);
        fclose(Output_fp);
    }

}
```

這邊是OSMutexPost，主要就新增printf一些訊息，一樣用EventName判斷當前的mutex是R1還是R2，接著比較MUTEX的優先權跟當前優先權，如果mutex的優先權大於當前Task的話，就printf mutex優先權 to Task優先權，反過來，如果任務的優先權大於當前Task的話，優先權不會改動，所以是 Task優先權 to Task優先權。

```
R1 = OSMutexCreate(R1MAXPrio, &err,1);
R2 = OSMutexCreate(R2MAXPrio, &err,2);
```

```
if (OSTCBCur->OSTCBR1UnLock != 0 || OSTCBCur->OSTCBR2UnLock != 0) {

    if (OSTCBCur->OSTCBWorkTime == OSTCBCur->OSTCBR1Lock && r1flag == 0 && OSTCBCur->OSTCBR1UnLock!=0) {
        OSMutexPend(R1, 0, &err);


        r1flag = 1;
    }
    if (OSTCBCur->OSTCBWorkTime == OSTCBCur->OSTCBR1UnLock && r1flag == 1) {

        r1flag = 0;

        OSMutexPost(R1);
        OS_Sched();

    }
    if (OSTCBCur->OSTCBWorkTime == OSTCBCur->OSTCBR2Lock && r2flag == 0 && OSTCBCur->OSTCBR2UnLock != 0) {
        OSMutexPend(R2, 0, &err);

        r2flag = 1;
    }
    if (OSTCBCur->OSTCBWorkTime == OSTCBCur->OSTCBR2UnLock && r2flag == 1) {

        r2flag = 0;

        OSMutexPost(R2);
        OS_Sched();

    }

}
```

最後在main裡面，要記得CreateMutex，裡面放的是優先權、err、跟自己加上去的name，然後Task裡面，綠框是判斷要使用mutex的時間點，時間到就Pend或Post，CPP就到這邊結束，紅框的Sched是為了Post之後可以立即排程。

[PART III] Performance Analysis [10%]

● Compare the scheduling behaviors between NPCS and CPP with PART I and PART II results. (5%)

NPCS在資源鎖定時就不讓其他Task有機會可以 preempt ，因此不會有太多的問題，但如果有其他Task優先權較高，且沒有使用到相同資源或不使用資源的話，也要等當前的資源被釋放才能被排程，簡單來說就是反應的時間會變慢；CPP會在資源鎖定的時候拉高Task的優先權到Mutex的優先權，而Mutex的優先權又會比所有有使用到的Task中最高的，因此如果其他Task的優先權比Mutex高的話，就能夠Preemption當前的Task(因為比Mutex高代表這個Task沒有使用這個Mutex)，因此反應的時間就會比NPCS快很多。

● Explain how NPCS and CPP avoid the deadlock problem. (5%)

NPCS 比較簡單，一拿到資源就不讓其他 Task preempt，因此其他 Task 不可能有機會拿走當前任務需要的資源，所以不會發生 deadlock。

CPP 的話，掌握資源的 Task 其優先權會被拉高到 Mutex，而 Mutex 的優先權會設定的比所有需要這個資源的 Task 還高，因此其他要用這個 Mutex 的也沒辦法 preempt，所以不會發生 deadlock。