嵌入式作業系統實作

Embedded OS Implementation

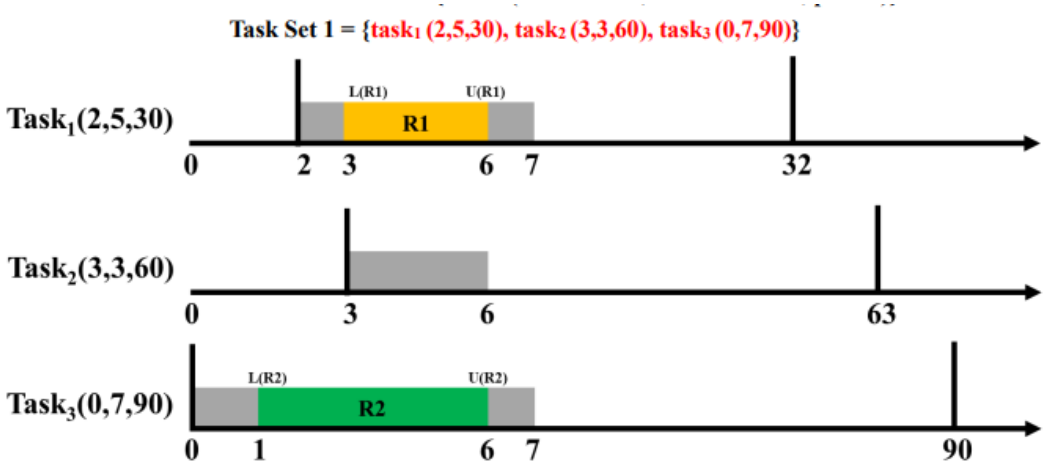
PA_3

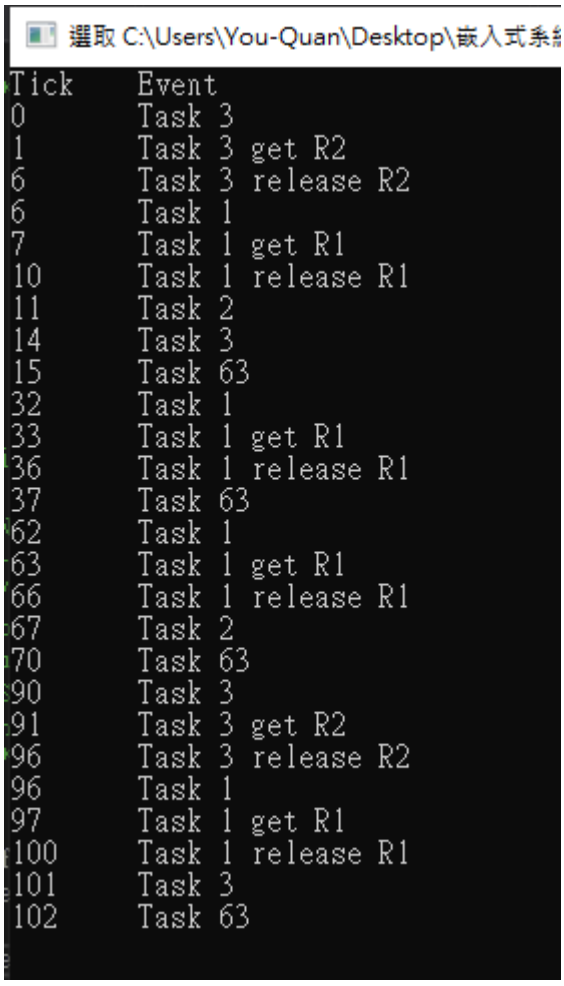
指導教授: 陳雅淑 教授

課程學生: M10907314 張祐銓

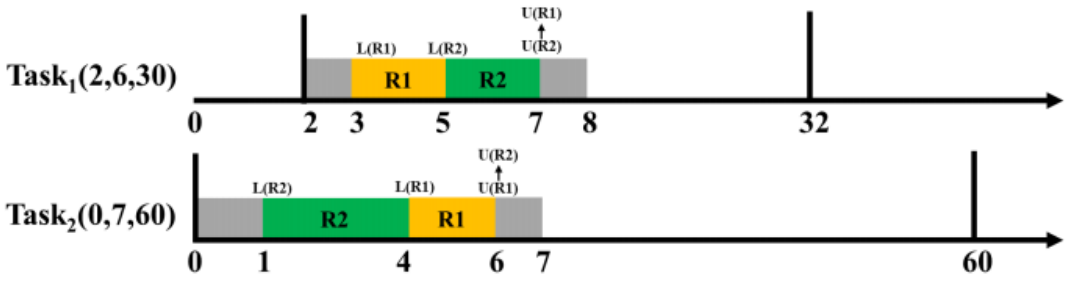
[PART I] NPCS Implementation [50%]

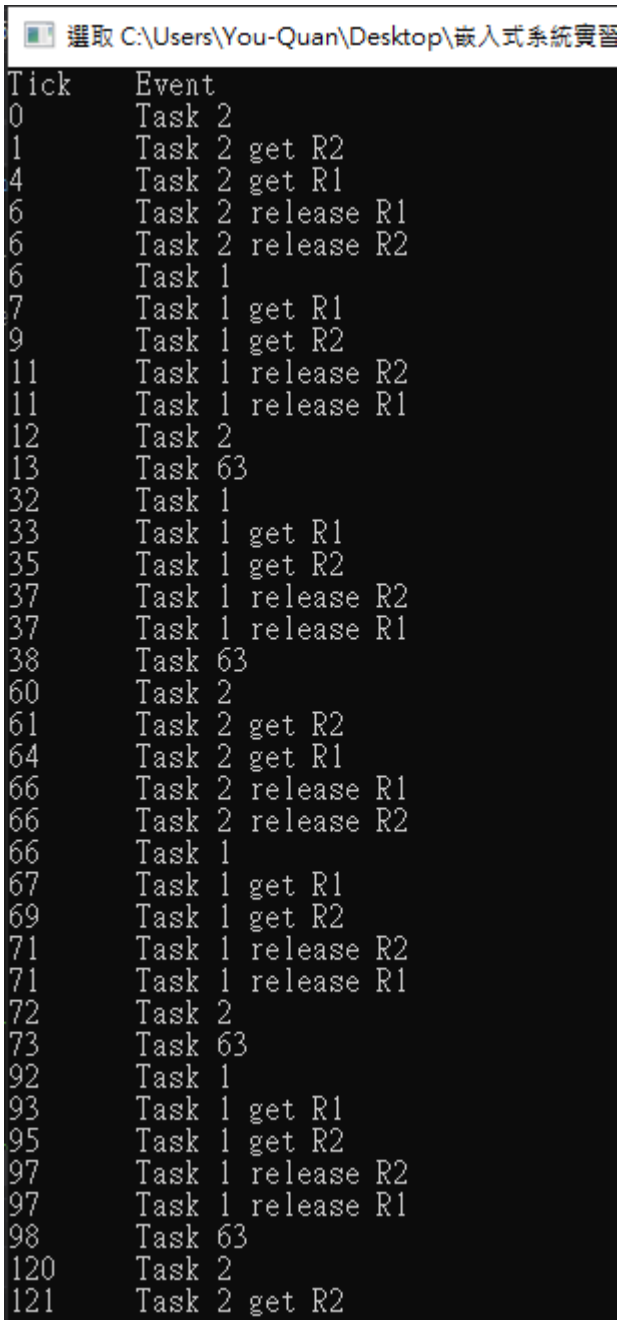
- The screenshot result (with the given format) of the two task sets. (Time tick 0-100) (10%)



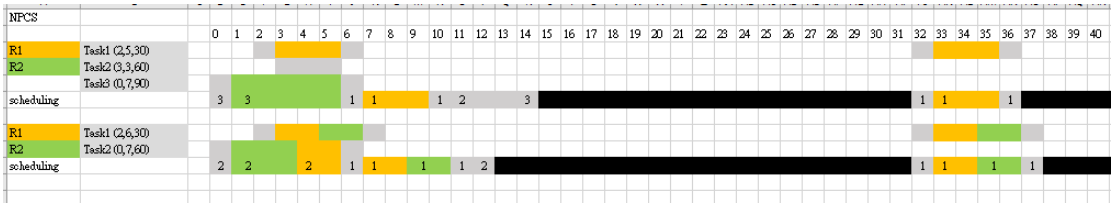


Task Set 2 = {task1 (2,6,30), task2 (0,7,60)}





- A report that describes your implementation, including scheduling results of two task sets, modified functions, data structure, etc. (please ATTACH the screenshot of the code and MARK the modified part). (40%)



上次 RMS 的實現條列說明:

1. 在OS_TCB中新增變數

```
INT32U begin_ready_time; //紀錄task變ready的時間點
INT32U response; //反應時間
INT32U arrival; //到達時間
INT32U execution; //執行時間
INT32U period; //週期
INT32U job_id; //工作次數
```

2. 在OSTimeTick中去察看目前的ReadyTable有哪些Task正在Ready狀態，把所有正在Ready且不是目前正在執行的task的所有task的response都加一，因為task正在Ready卻沒有被執行表示被延後了一個tick，response一開始初始值是週期內的執行時間，所以task工作執行只要沒有執行滿response的時間就會一直卡在while迴圈內。

3. Task執行完會執行OS_Sched()去切換給下個Task，所以在OS_Sched()中顯示完成的字串。

4. 當TimeTick中斷產生之後會進OSIntExit()把低優先權的task中斷給高優先權的task，所以在這裡顯示中斷狀態的字串。

5. 當有task已經快要完成時有可能會被其他優先權高的task給搶占，所以會在OSIntExit()增加判斷程式迴避掉那次的context switch，讓快要做完的task先完成它的工作。

6. 當有一個task完成了一個周期內的工作後要做下個週期的工作時因為不會進入OS_Sched()做context switch，所以在OSTimeDly(0)時表示task的反應時間等於週期，在OSTimeDly內去print完成的字串。

7. 判斷是否有task已經miss deadline，只要在OSTimeTick中去判斷reponse是否大於period，表示說反應時間已經大於週期就是miss deadline。

NPCS 的實現條列說明:

1. 在OS_TCB中新增變數

```
INT32U deadline; //EDF的最後期限
INT32U runtime; //執行了多久
```

2. 在 ucos_ii.h 中新增 OS_EXT OS_EVENT *Jack_event_Tbl[OS_LOWEST_PRIO + 1u] ，記錄所有 mutex event。

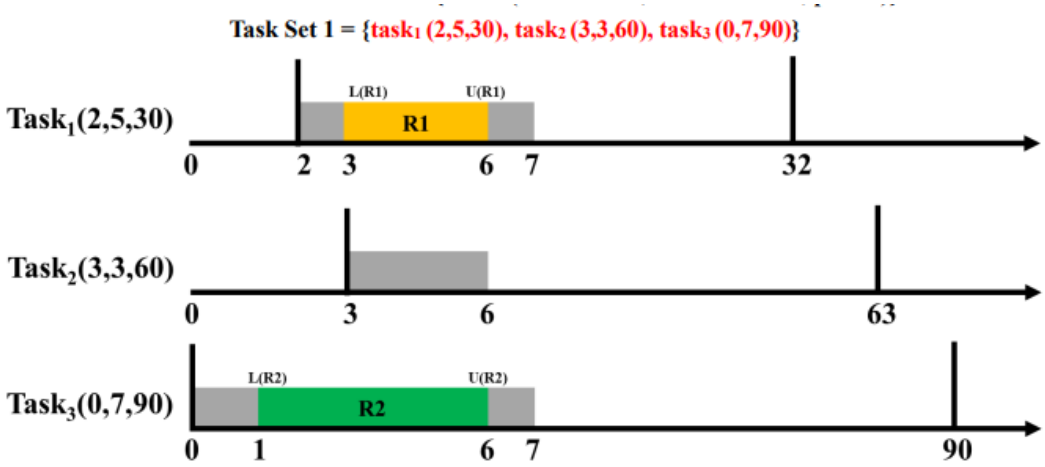
3. 在 OSTimeTick()中當前工作的 task 的 runtime 就加 1。

4. 在 OSIntExit()和 OS_Sched()中會做 context switch，在做之前先做判斷，判斷 Jack_event_Tbl 中當前 task 是否有占用任何資源，若有占用到資源就迴避不做 context switch。

- 程式截圖部分跟 CPP 的程式截圖整合在一起

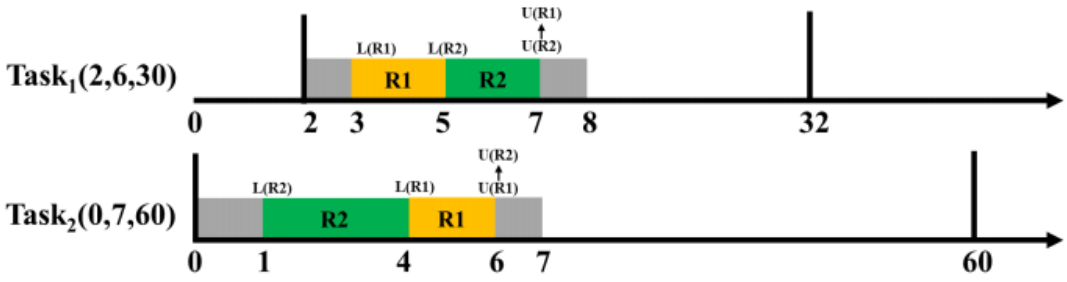
[PART II] CPP Implementation [40%]

- The screenshot result (with the given format) of the two task sets. (Time tick 0-100) (10%)




```
選取 C:\Users\You-Quan\Desktop\嵌入式系統實習\PA3\Micrium_Win3
Tick    Event                   Prio_Inheritance
0       Task 3
1       Task 3 get R2           5>>4
2       Task 1
3       Task 1 get R1           2>>1
6       Task 1 release R1       1>>2
7       Task 2
10      Task 3
14      Task 3 release R2       4>>5
15      Task 63
32      Task 1
33      Task 1 get R1           2>>1
36      Task 1 release R1       1>>2
37      Task 63
62      Task 1
63      Task 1 get R1           2>>1
66      Task 1 release R1       1>>2
67      Task 2
70      Task 63
90      Task 3
91      Task 3 get R2           5>>4
92      Task 1
93      Task 1 get R1           2>>1
96      Task 1 release R1       1>>2
97      Task 3
101     Task 3 release R2       4>>5
102     Task 63
```

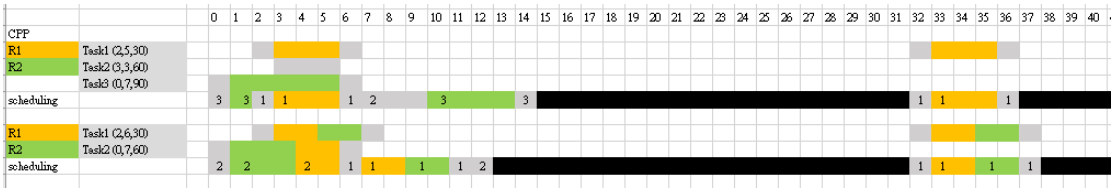
Task Set 2 = {task1 (2,6,30), task2 (0,7,60)}



選取 C:\Users\You-Quan\Desktop\嵌入式系統實習\PA3\Micrium_Wir

Tick	Event	Prio_Inheritance
0	Task 2	
1	Task 2 get R2	4>>2
4	Task 2 get R1	2>>1
6	Task 2 release R1	1>>2
6	Task 2 release R2	2>>4
6	Task 1	
7	Task 1 get R1	3>>1
9	Task 1 get R2	1>>1
11	Task 1 release R2	1>>1
11	Task 1 release R1	1>>3
12	Task 2	
13	Task 63	
32	Task 1	
33	Task 1 get R1	3>>1
35	Task 1 get R2	1>>1
37	Task 1 release R2	1>>1
37	Task 1 release R1	1>>3
38	Task 63	
60	Task 2	
61	Task 2 get R2	4>>2
64	Task 2 get R1	2>>1
66	Task 2 release R1	1>>2
66	Task 2 release R2	2>>4
66	Task 1	
67	Task 1 get R1	3>>1
69	Task 1 get R2	1>>1
71	Task 1 release R2	1>>1
71	Task 1 release R1	1>>3
72	Task 2	
73	Task 63	
92	Task 1	
93	Task 1 get R1	3>>1
95	Task 1 get R2	1>>1
97	Task 1 release R2	1>>1
97	Task 1 release R1	1>>3
98	Task 63	
120	Task 2	
121	Task 2 get R2	4>>2

- A report that describes your implementation, including scheduling results of two task sets, modified functions, data structure, etc. (please ATTACH the screenshot of the code and MARK the modified part). (30%)



CPP 實現方式條列說明:

1. 根據 NPCS 再進行後續的追加實現 CPP。
2. 在 OSIntExit()和 OS_Sched()中會做 context switch，在做之前先做判斷，判斷 Jack_event_Tbl 中當前 task 占用的資源中，若有占用的資源的優先權高於下個要切換的 task 的話，就迴避不做這次的 context switch。

Main.c

```
61      static  OS_STK  StartupTaskStk[APP_CFG_STARTUP_TASK_STK_SIZE];
62
63      #define TASK_STACKSIZE 2048
64
65      static OS_STK Task1_STK[TASK_STACKSIZE];
66      static OS_STK Task2_STK[TASK_STACKSIZE];
67      static OS_STK Task3_STK[TASK_STACKSIZE];
68
69      #define TASK1_ID 1 //定義task1的id、priority、arrival、execution、period
70      #define TASK1_PRIORITY 3
71      #define TASK1_ARRIVAL 2
72      #define TASK1_EXECUTION 6
73      #define TASK1_PERIOD 30
74
75      #define TASK2_ID 2 //定義task2的id、priority、arrival、execution、period
76      #define TASK2_PRIORITY 4
77      #define TASK2_ARRIVAL 0
78      #define TASK2_EXECUTION 7
79      #define TASK2_PERIOD 60
80
81      /*#define TASK3_ID 3 //定義task3的id、priority、arrival、execution、period
82      #define TASK3_PRIORITY 5
83      #define TASK3_ARRIVAL 0
84      #define TASK3_EXECUTION 7
85      #define TASK3_PERIOD 90*/
86
87      //R1的ID為1   R2的ID為2
88      #define R1_ID 1
89      #define R1_PRIO 1
90      #define R2_ID 2
91      #define R2_PRIO 2
92
93      OS_EVENT* R1;
94      OS_EVENT* R2;
```

```
#ifdef TASK1_ID //假如task1有被定義的話 就創建task1工作
    OSTaskCreateExt(task1,
        0,
        &Task1_STK[TASK_STACKSIZE - 1u],
        TASK1_PRIORITY,
        TASK1_ID,
        &Task1_STK[0u],
        TASK_STACKSIZE,
        0u,
        (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR),
        TASK1_ARRIVAL,
        TASK1_EXECUTION,
        TASK1_PERIOD);
#endif // TASK1_ID


#ifdef TASK2_ID //假如task2有被定義的話 就創建task2工作
    OSTaskCreateExt(task2,
        0,
        &Task2_STK[TASK_STACKSIZE - 1u],
        TASK2_PRIORITY,
        TASK2_ID,
        &Task2_STK[0u],
        TASK_STACKSIZE,
        0u,
        (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR),
        TASK2_ARRIVAL,
        TASK2_EXECUTION,
        TASK2_PERIOD);
#endif // TASK2_ID
```

```
#ifdef TASK3_ID //假如task3有被定義的話 就創建task3工作
    OSTaskCreateExt(task3,
        0,
        &Task3_STK[TASK_STACKSIZE - 1u],
        TASK3_PRIORITY,
        TASK3_ID,
        &Task3_STK[0u],
        TASK_STACKSIZE,
        0u,
        (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR),
        TASK3_ARRIVAL,
        TASK3_EXECUTION,
        TASK3_PERIOD);
#endif // TASK3_ID

    INT8U err;
    R1 = Jack_OSMutexCreate(R1_PRIO, &err, R1_ID);
    R2 = Jack_OSMutexCreate(R2_PRIO, &err, R2_ID);

    OSTimeSet(0); //重新歸零timetick
```

```
250  void mywait(int tick)
251  {
252  #if OS_CRITICAL_METHOD == 3
253      OS_CPU_SR cpu_sr = 0;
254  #endif
255
256      int now, exit;
257      OS_ENTER_CRITICAL();
258      now = OSTCBCur->runtime; //now 為目前task執行多久的時間點
259      exit = now + tick; //exit 為now加等待多久tick
260      OS_EXIT_CRITICAL();
261      while (1)
262      {
263          //OSTCBCur->runtime會在只有在當前task在執行時才會加時間
264          //exit只要小於等於runtime時就會跳出迴圈
265          if (exit <= OSTCBCur->runtime)
266              break;
267      }
268  }
```

```
270 void task1(void* p_arg) //task1的執行程式
271 {
272     (void)p_arg;
273     INT8U err;
274
275     while (1)
276     {
277         //假如task開始執行時 還沒到達arrival time那就執行 OSTimeDly休息相差的時間
278         if (OSTCBCur->arrival > OSTimeGet())
279         {
280             OSTimeDly(OSTCBCur->arrival - OSTimeGet());
281         }
282         else
283         {
284             while (1)
285             {
286                 //第一小題的題目設定
287                 /*mywait(1);
288
289                 OSMutexPend(R1, 0, &err);
290                 mywait(3);
291
292                 OSMutexPost(R1);
293                 mywait(1);
294
295                 //printf("task 1 sleep %d\n", OSTCBCur->deadline - OSTimeGet());
296                 OSTimeDly(OSTCBCur->deadline - OSTimeGet());*/
297
298
299                 //第二小題的題目設定
300                 mywait(1);
301
302                 OSMutexPend(R1, 0, &err);
303                 mywait(2);
304
305                 OSMutexPend(R2, 0, &err);
306                 mywait(2);
307
308                 OSMutexPost(R2);
309                 OSMutexPost(R1);
310                 mywait(1);
311
312                 OSTimeDly(OSTCBCur->deadline - OSTimeGet());
313             }
314         }
315     }
316 }
```

```
318 void task2(void* p_arg) //task2的執行程式
319 {
320     (void)p_arg;
321     INT8U err;
322 
323     while (1)
324     {
325         //假如task開始執行時 還沒到達arrival time那就執行 OSTimeDly休息相差的時間
326         if (OSTCBCur->arrival > OSTimeGet())
327         {
328             OSTimeDly(OSTCBCur->arrival - OSTimeGet());
329         }
330         else
331         {
332 
333             while (1)
334             {
335                 //第一小題的題目設定
336                 /*mywait(3);
337 
338                 OSTimeDly(OSTCBCur->deadline - OSTimeGet());*/
339 
340                 //第二小題的題目設定
341                 mywait(1);
342 
343                 OSMutexPend(R2, 0, &err);
344                 mywait(3);
345 
346                 OSMutexPend(R1, 0, &err);
347                 mywait(2);
348 
349                 OSMutexPost(R1);
350                 OSMutexPost(R2);
351                 mywait(1);
352 
353                 OSTimeDly(OSTCBCur->deadline - OSTimeGet());
354 
355             }
356         }
357     }
358 }
```

```
360  void task3(void* p_arg) //task3的執行程式
361  {
362      (void)p_arg;
363      INT8U err;
364
365      while (1)
366      {
367          //假如task開始執行時 還沒到達arrival time那就執行 OSTimeDly休息相差的時間
368          if (OSTCBCur->arrival > OSTimeGet())
369          {
370              OSTimeDly(OSTCBCur->arrival - OSTimeGet());
371          }
372          else
373          {
374
375              while (1)
376              {
377                  //第一小題的題目設定
378                  mywait(1);
379
380                  OSMutexPend(R2, 0, &err);
381                  mywait(5);
382
383                  OSMutexPost(R2);
384                  mywait(1);
385
386                  OSTimeDly(OSTCBCur->deadline - OSTimeGet());
387
388              }
389          }
390      }
391  }
392
```

ucos_ii.h

```
639      //在OS_TCB內新增變數
640
641      INT32U begin_ready_time; //紀錄task變為ready的時間點
642      INT32U response; //反應時間
643      INT32U arrival; //到達時間
644      INT32U execution; //執行時間
645      INT32U period; //週期
646      INT32U job_id; //工作次數
647      INT32U deadline; //最後期限
648      INT32U runtime; //執行了多久
649  } OS_TCB;
650
```

```
711   OS_EXT  INT32U           OSCtxSwCtr;              /* Counter of number of context switches          */
712
713   #if (OS_EVENT_EN) && (OS_MAX_EVENTS > 0u)
714   OS_EXT  OS_EVENT        *OSEventFreeList;         /* Pointer to list of free EVENT control blocks   */
715   OS_EXT  OS_EVENT         OSEventTbl[OS_MAX_EVENTS];/* Table of EVENT control blocks                  */
716   OS_EXT  OS_EVENT        *Jack_event_Tbl[OS_LOWEST_PRIO + 1u];    //新增一個Mutex event的table
717   #endif
```

```
386   #if (OS_EVENT_EN) && (OS_MAX_EVENTS > 0u)
387   typedef struct os_event {
388       INT8U    OSEventType;                 /* Type of event control block (see OS_EVENT_TYPE_xxxx)    */
389       void    *OSEventPtr;                  /* Pointer to message or queue structure                   */
390       INT16U   OSEventCnt;                  /* Semaphore Count (not used if other EVENT type)          */
391       OS_PRIO  OSEventGrp;                  /* Group corresponding to tasks waiting for event to occur */
392       OS_PRIO  OSEventTbl[OS_EVENT_TBL_SIZE]; /* List of tasks waiting for event to occur              */
393       INT8U    id;//紀錄resource的id  ex:R1的ID為1、R2的ID為2
394
395   #if OS_EVENT_NAME_EN > 0u
396       INT8U   *OSEventName;
397   #endif
398   } OS_EVENT;
399   #endif
400
```

os_cfg.h

```
25    #ifndef OS_CFG_H
26    #define OS_CFG_H
27
28    #define CPP  //設定要用什麼樣的排程方式  #define CPP 或 #define NPCS
29                                            /* ---------------------- MISCELLANEOUS ----------------------- */
30    #define OS_APP_HOOKS_EN           1u    /* Application-defined hooks are called from the uC/OS-II hooks */
31    #define OS_ARG_CHK_EN             1u    /* Enable (1) or Disable (0) argument checking                  */
32    #define OS_CPU_HOOKS_EN           1u    /* uC/OS-II hooks are found in the processor port files         */
33
34    #define OS_DEBUG_EN               1u    /* Enable(1) debug variables                                    */
35
36    #define OS_EVENT_MULTI_EN         1u    /* Include code for OSEventPendMulti()                          */
37    #define OS_EVENT_NAME_EN          1u    /* Enable names for Sem, Mutex, Mbox and Q                      */
38
39    #define OS_LOWEST_PRIO           63u    /* Defines the lowest priority that can be assigned ...         */
40                                            /* ... MUST NEVER be higher than 254!                           */
41
42    #define OS_MAX_EVENTS            10u    /* Max. number of event control blocks in your application      */
43    #define OS_MAX_FLAGS              5u    /* Max. number of Event Flag Groups    in your application      */
44    #define OS_MAX_MEM_PART           5u    /* Max. number of memory partitions                             */
45    #define OS_MAX_QS                 4u    /* Max. number of queue control blocks in your application      */
46    #define OS_MAX_TASKS             20u    /* Max. number of tasks in your application, MUST be >= 2       */
47
48    #define OS_SCHED_LOCK_EN          1u    /* Include code for OSSchedLock() and OSSchedUnlock()           */
49
50    #define OS_TICK_STEP_EN           1u    /* Enable tick stepping feature for uC/OS-View                  */
51    #define OS_TICKS_PER_SEC          1u    /* Set the number of ticks in one second                        */
52
53    #define OS_TLS_TBL_SIZE           0u    /* Size of Thread-Local Storage Table                           */
54
```

os_task. c

```
349  INT8U  OSTaskCreateExt (void   (*task)(void *p_arg),
350                          void    *p_arg,
351                          OS_STK  *ptos,
352                          INT8U    prio,
353                          INT16U   id,
354                          OS_STK  *pbos,
355                          INT32U   stk_size,
356                          void    *pext,
357                          INT16U   opt,
358                          INT32U   arrival,
359                          INT32U   execution,
360                          INT32U   period) //在OSTaskCreateExt內新增變數 arrival、execution、period
361  {
362      OS_STK     *psp;
363      INT8U       err;
364  #if OS_CRITICAL_METHOD == 3u                 /* Allocate storage for CPU status register             */
365      OS_CPU_SR   cpu_sr = 0u;
366  #endif
367
368
369
370  #ifdef OS_SAFETY_CRITICAL_IEC61508
371      if (OSSafetyCriticalStartFlag == OS_TRUE) {
372          OS_SAFETY_CRITICAL_EXCEPTION();
373          return (OS_ERR_ILLEGAL_CREATE_RUN_TIME);
374      }
375  #endif
376
377  #if OS_ARG_CHK_EN > 0u
378      if (prio > OS_LOWEST_PRIO) {             /* Make sure priority is within allowable range         */
379          return (OS_ERR_PRIO_INVALID);
380      }
381  #endif
382      OS_ENTER_CRITICAL();
383      if (OSIntNesting > 0u) {                 /* Make sure we don't create the task from within an ISR */
384          OS_EXIT_CRITICAL();
385          return (OS_ERR_TASK_CREATE_ISR);
386      }
387      if (OSTCBPrioTbl[prio] == (OS_TCB *)0) { /* Make sure task doesn't already exist at this priority  */
388          OSTCBPrioTbl[prio] = OS_TCB_RESERVED;/* Reserve the priority to prevent others from doing ...  */
389                                               /* ... the same thing until task is created.              */
390          OS_EXIT_CRITICAL();
391
392  #if (OS_TASK_STAT_STK_CHK_EN > 0u)
393          OS_TaskStkClr(pbos, stk_size, opt);                  /* Clear the task stack (if needed)     */
394  #endif
395
396          psp = OSTaskStkInit(task, p_arg, ptos, opt);         /* Initialize the task's stack          */
397          err = OS_TCBInit(prio, psp, pbos, id, stk_size, pext, opt, arrival, execution, period); //在OS_TCBInit內新增變數 arrival、execution、period
```

os_core. c

```
938  void  OSStart (void)
939  {
940      if (OSRunning == OS_FALSE) {
941
942  #ifdef NPCS
943          printf("Tick\tEvent\n");
944  #endif //  NPCS
945
946  #ifdef CPP
947          printf("Tick\tEvent\t\t\tPrio_Inheritance\n");
948  #endif // CPP
949
950          OS_SchedNew();                               /* Find highest priority's task priority number   */
951          OSPrioCur     = OSPrioHighRdy;
952          OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy]; /* Point to highest priority task ready to run    */
953          OSTCBCur      = OSTCBHighRdy;
954          OSStartHighRdy();                            /* Execute target specific code to start task     */
955      }
956  }
```

```
2231    INT8U  OS_TCBInit (INT8U    prio,
2232                       OS_STK  *ptos,
2233                       OS_STK  *pbos,
2234                       INT16U   id,
2235                       INT32U   stk_size,
2236                       void    *pext,
2237                       INT16U   opt,
2238                       INT32U   arrival,
2239                       INT32U   execution,
2240                       INT32U   period) //在OS_TCBInit內新增變數 arrival、execution、period
2241    {
2242        OS_TCB    *ptcb;
2243    #if OS_CRITICAL_METHOD == 3u                          /* Allocate storage for CPU status register */
2244        OS_CPU_SR  cpu_sr = 0u;
2245    #endif
2246    #if OS_TASK_REG_TBL_SIZE > 0u
2247        INT8U      i;
2248    #endif
2249    #if OS_TASK_CREATE_EXT_EN > 0u
2250    #if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
2251        INT8U      j;
2252    #endif
2253    #endif
2254
2255
2256        OS_ENTER_CRITICAL();
2257        ptcb = OSTCBFreeList;                             /* Get a free TCB from the free TCB list    */
2258        if (ptcb != (OS_TCB *)0) {
2259            OSTCBFreeList           = ptcb->OSTCBNext;    /* Update pointer to free TCB list          */
2260            OS_EXIT_CRITICAL();
2261
2262            ptcb->begin_ready_time = arrival; //初始化在OS_TCB內新增的變數
2263            ptcb->response = execution;
2264            ptcb->arrival = arrival;
2265            ptcb->execution = execution;
2266            ptcb->period = period;
2267            ptcb->job_id = 0;
2268            ptcb->deadline = arrival + period;
2269            ptcb->runtime = 0;
```

OSTimeTick()

```
1061        //這段是加在OSTimeTick內
1062        //從ReadyTable得到目前正在Ready的task
1063        //把現在正在Ready但卻沒有在執行的task的反應時間加一
1064        for (INT8U y = 0; y < 8; y++) //jack
1065        {
1066            for (INT8U x = 0; x < 8; x++)
1067            {
1068
1069                if ((OSRdyTbl[y] & (INT8U)1 << x) == (INT8U)1 << x) //確認是有有Ready
1070                {
1071                    INT8U prio = (y << 3u) + x;
1072                    if (prio != OS_TASK_IDLE_PRIO)
1073                    {
1074                        if (OSTCBPrioTbl[prio] != OSTCBCur && OSTimeGet() > OSTCBPrioTbl[prio]->arrival)
1075                        {
1076                            OSTCBPrioTbl[prio]->response += 1;
1077
1078                            //假設反應時間 大於週期 表示說task已經missdeadline了
1079                            if (OSTCBPrioTbl[prio]->response > OSTCBPrioTbl[prio]->period)
1080                            {
1081                                printf("%d\t MissDeadline\t Task(%d)(%d)\t\t ----------\n", OSTimeGet(),
1082                                    OSTCBPrioTbl[prio]->OSTCBId, OSTCBPrioTbl[prio]->job_id);
1083                                while (1) {}
1084                            }
1085                        }
1086
1087                        //只要是面前正在執行的task，runtime加1
1088                        if (OSTCBPrioTbl[prio] == OSTCBCur)
1089                        {
1090                            OSTCBCur->runtime += 1;
1091                        }
1092                    }
1093                }
1094            }
1095        }
1096
```

OSTimeTick()

```
1112        ptcb = OSTCBList;                                      /* Point at first TCB in TCB list               */
1113        while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {         /* Go through all TCBs in TCB list              */
1114            OS_ENTER_CRITICAL();
1115            if (ptcb->OSTCBDly != 0u) {                        /* No, Delayed or waiting for event with TO     */
1116                ptcb->OSTCBDly--;                              /* Decrement nbr of ticks to end of delay       */
1117                if (ptcb->OSTCBDly == 0u) {                    /* Check for timeout                            */
1118
1119                    if ((ptcb->OSTCBStat & OS_STAT_PEND_ANY) != OS_STAT_RDY) {
1120                        ptcb->OSTCBStat  &= (INT8U)~(INT8U)OS_STAT_PEND_ANY;   /* Yes, Clear status flag   */
1121                        ptcb->OSTCBStatPend = OS_STAT_PEND_TO;                 /* Indicate PEND timeout    */
1122                    } else {
1123                        ptcb->OSTCBStatPend = OS_STAT_PEND_OK;
1124                    }
1125
1126                    if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == OS_STAT_RDY) {  /* Is task suspended?       */
1127                        OSRdyGrp               |= ptcb->OSTCBBitY;             /* No,  Make ready          */
1128                        OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
1129
1130                        //這段是加在OSTimeTick內
1131                        //當某個task轉變為Ready的時間 設定變數數值
1132                        ptcb->begin_ready_time = OSTimeGet();  //begin_ready_time是task轉變為Ready的時間點
1133                        ptcb->response = ptcb->execution;
1134                        ptcb->OSTCBCtxSwCtr = 0; //context switch次數設為0
1135                        ptcb->deadline = (ptcb->job_id + 1) * ptcb->period + ptcb->arrival; //jack
1136                        ptcb->runtime = 0;
1137
1138                        OS_TRACE_TASK_READY(ptcb);
1139                    }
1140                }
1141            }
1142            ptcb = ptcb->OSTCBNext;                            /* Point at next TCB in TCB list                */
1143            OS_EXIT_CRITICAL();
1144        }
1145    }
1146)
1147
```

```
1839  void  OS_Sched (void)
1840  {
1841  #if OS_CRITICAL_METHOD == 3u                       /* Allocate storage for CPU status register
1842      OS_CPU_SR  cpu_sr = 0u;
1843  #endif
1844
1845
1846      OS_ENTER_CRITICAL();
1847      if (OSIntNesting == 0u) {                      /* Schedule only if all ISRs done and ...
1848          if (OSLockNesting == 0u) {                 /* ... scheduler is not locked
1849              OS_SchedNew();
1850              OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
1851
1852              INT8U sw = 1; //紀錄是否要context switch
1853
1854              //遍歷所有Jack_event_Tbl表
1855              //查看是否有當前task使用的資源的優先權比OSTCBHighRdy還高
1856              for (int i = 0; i < 64; i++)
1857              {
1858                  if (Jack_event_Tbl[i] != (OS_EVENT*)0)
1859                  {
1860                      if ((Jack_event_Tbl[i]->OSEventCnt & 0x00FF) == OSPrioCur)
1861                      {
1862
1863  #ifdef NPCS
1864                          //排程為NPCS時執行這段
1865                          //只要有使用任一資源就不做context switch
1866                          if ((Jack_event_Tbl[i]->OSEventCnt >> 8) != 0x00)
1867                          {
1868                              sw = 0;
1869                              OSPrioHighRdy = OSPrioCur;
1870                              OSTCBHighRdy = OSTCBCur;
1871                          }
1872  #endif // NPCS
1873  #ifdef CPP
1874                          //排程為CPP時執行這段
1875                          //只要當前task使用的資源的優先權比OSTCBHighRdy還高就不做context switch
1876                          if ((Jack_event_Tbl[i]->OSEventCnt >> 8) <= OSPrioHighRdy)
1877                          {
1878                              sw = 0;
1879                              OSPrioHighRdy = OSPrioCur;
1880                              OSTCBHighRdy = OSTCBCur;
1881                          }
1882  #endif // CPP
1883                      }
1884                  }
1885              }
1886
```

```
1887              if (sw == 1) //是否做context switch
1888              {
1889                  if (OSPrioHighRdy != OSPrioCur) {          /* No Ctx Sw if current task is highest rdy     */
1890
1891                      //顯示下個要執行的task
1892                      if (OSPrioHighRdy == OS_TASK_IDLE_PRIO)
1893                          printf("%d\tTask %d\n", OSTimeGet(), OS_TASK_IDLE_PRIO);
1894                      else if (OSTimeGet() >= OSTCBHighRdy->arrival)
1895                          printf("%d\tTask %d\n", OSTimeGet(), OSTCBHighRdy->OSTCBId);
1896
1897                      if (OSTimeGet() > OSTCBCur->arrival)
1898                      {
1899                          OSTCBCur->OSTCBCtxSwCtr++; //OSTCBCtxSwCtr加一
1900
1901
1902                          //task完成之後會執行OS_Sched所以在這裡print完成
1903                          //假設是task完成後切入idle task，print的方式會不一樣
1904                          /*if(OSTCBHighRdy->OSTCBPrio == OS_TASK_IDLE_PRIO)
1905                              printf("%d\t Completion\t task(%d)(%d)\t\t task(%d)  \t\t %d\t\t\t\t %d\n", OSTimeGet(),
1906                                  OSTCBCur->OSTCBId, OSTCBCur->job_id,
1907                                  OS_TASK_IDLE_PRIO,
1908                                  OSTimeGet() - OSTCBCur->begin_ready_time, OSTCBCur->OSTCBCtxSwCtr);
1909                          else
```

OS_Sched()

```
696  void  OSIntExit (void)
697  {
698  #if OS_CRITICAL_METHOD == 3u                          /* Allocate storage for CPU status register */
699      OS_CPU_SR  cpu_sr = 0u;
700  #endif
701
702      if (OSRunning == OS_TRUE) {
703          OS_ENTER_CRITICAL();
704          if (OSIntNesting > 0u) {                      /* Prevent OSIntNesting from wrapping       */
705              OSIntNesting--;
706          }
707          if (OSIntNesting == 0u) {                     /* Reschedule only if all ISRs complete ... */
708              if (OSLockNesting == 0u) {                /* ... and not locked.                      */
709
710                  //加這行if判斷是為了迴避當有task已經快要完成時被其他優先權高的task給搶占
711                  //迴避掉這次切換讓原本的task先做完
712                  if (OSTimeGet() - OSTCBCur->begin_ready_time != OSTCBCur->response)
713                  {
714                      OS_SchedNew();
715                      OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
716
717                      INT8U sw = 1; //紀錄是否要context switch
718
719                      //遍歷所有Jack_event_Tbl表
720                      //查看是否有當前task使用的資源的優先權比OSTCBHighRdy還高
721                      for (int i = 0; i < 64; i++)
722                      {
723                          if (Jack_event_Tbl[i] != (OS_EVENT*)0)
724                          {
725                              if ((Jack_event_Tbl[i]->OSEventCnt & 0x00FF) == OSPrioCur)
726                              {
727  #ifdef NPCS
728                                  //排程為NPCS時執行這段
729                                  //只要有使用任一資源就不做context switch
730                                  if ((Jack_event_Tbl[i]->OSEventCnt >> 8) != 0x00)
731                                  {
732                                      sw = 0;
733                                      OSPrioHighRdy = OSPrioCur;
734                                      OSTCBHighRdy = OSTCBCur;
735                                  }
736  #endif // NPCS
737  #ifdef CPP
738                                  //排程為CPP時執行這段
739                                  //只要當前task使用的資源的優先權比OSTCBHighRdy還高就不做context switch
740                                  if ((Jack_event_Tbl[i]->OSEventCnt >> 8) <= OSPrioHighRdy)
741                                  {
742                                      sw = 0;
743                                      OSPrioHighRdy = OSPrioCur;
744                                      OSTCBHighRdy = OSTCBCur;
745                                  }
```

```
751                  if(sw == 1) //確認是否要做context switch
752                  {
753                  if (OSPrioHighRdy != OSPrioCur) {          /* No Ctx Sw if current task is highest rdy */
754
755                      //顯示下個要執行的task
756                      if (OSPrioHighRdy == OS_TASK_IDLE_PRIO)
757                          printf("%d\tTask %d\n", OSTimeGet(), OS_TASK_IDLE_PRIO);
758                      else
759                          printf("%d\tTask %d\n", OSTimeGet(), OSTCBHighRdy->OSTCBId);
760
761                      OSTCBCur->OSTCBCtxSwCtr++; //OSTCBCtxSwCtr加一
762
763                      //OSIntExit是在timetick中斷後會做，低優先權task會被高優先權task搶佔，所以在這裡print搶佔
764                      //假設是idle task被搶占，print的方式會不一樣
765                      /*if (OSTCBCur->OSTCBPrio == OS_TASK_IDLE_PRIO)
766                          printf("%d\t Preemption\t task(%d)  \t\t task(%d)(%d)\n", OSTimeGet(),
767                              OS_TASK_IDLE_PRIO,
768                              OSTCBHighRdy->OSTCBId, OSTCBHighRdy->job_id);
769                      else
770                          printf("%d\t Preemption\t task(%d)(%d)\t\t task(%d)(%d)\n", OSTimeGet(),
771                              OSTCBCur->OSTCBId, OSTCBCur->job_id,
772                              OSTCBHighRdy->OSTCBId, OSTCBHighRdy->job_id);*/
```

os_time.c

```
55  void  OSTimeDly (INT32U ticks)
56  {
57      INT8U      y;
58  #if OS_CRITICAL_METHOD == 3u                 /* Allocate storage for CPU status register           */
59      OS_CPU_SR  cpu_sr = 0u;
60  #endif
61
62      if (ticks > 0u && OSTCBCur->arrival < OSTimeGet())
63      {
64          //task初始化參數
65          OSTCBCur->begin_ready_time = OSTimeGet();
66          OSTCBCur->response = OSTCBCur->execution;
67          OSTCBCur->OSTCBCtxSwCtr = 0;
68
69          //工作次數+1
70          OSTCBCur->job_id += 1;
71
72          OSTCBCur->deadline = (OSTCBCur->job_id + 1) * OSTCBCur->period + OSTCBCur->arrival;
73      }
```

```
95      //當發生直行 OSTimeDly(0) 的情況表示反應時間剛好是週期時間
96      //task已經完成這次週期的工作要做下個週期的工作
97      if (OSTCBCur->period == OSTCBCur->response)
98      {
99          //print這次週期的工作完成 輪到下個週期的工作開始
100         /*printf("%d\t Completion\t task(%d)(%d)\t\t task(%d)(%d)\t\t %d\t\t\t %d\n", OSTimeGet(),
101             OSTCBCur->OSTCBId, OSTCBCur->job_id,
102             OSTCBCur->OSTCBId, OSTCBCur->job_id + 1,
103             OSTimeGet() - OSTCBCur->begin_ready_time, OSTCBCur->OSTCBCtxSwCtr);*/
104
105         //初始化task的OS_TCB參數
106         OSTCBCur->begin_ready_time = OSTimeGet();
107         OSTCBCur->response = OSTCBCur->execution;
108         OSTCBCur->OSTCBCtxSwCtr = 0;
109
110         //工作次數+1
111         OSTCBCur->job_id += 1;
112     }
113 }
114
```

OSTimeDly()

os_cpu_c.c

```
624
625 #if (OS_MSG_TRACE > 0u)
626     //OS_Printf("Task[%3.1d] created, Thread ID %5.0d\n", p_tcb->OSTCBPrio, p_stk->ThreadID); //因為作業不需要print這行 所以註解掉這行
627 #endif
```

os_mutex.c

```
    pevent = OSEventFreeList;                                  /* Get next free event control block        */
    if (pevent == (OS_EVENT*)0) {                              /* See if an ECB was available              */
        if (prio != OS_PRIO_MUTEX_CEIL_DIS) {
            OSTCBPrioTbl[prio] = (OS_TCB*)0;                   /* No, Release the table entry              */
        }
        OS_EXIT_CRITICAL();
        *perr = OS_ERR_PEVENT_NULL;                            /* No more event control blocks             */
        return (pevent);
    }
    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                 */
    pevent->Id = Id;
    Jack_event_Tbl[id] = pevent; //將mutex的記憶體位置放入Jack_event_Tbl中
#if OS_EVENT_NAME_EN > 0u
    pevent->OSEventName = (INT8U*)(void*)"?";
```

Jack_OSMutexCreate()

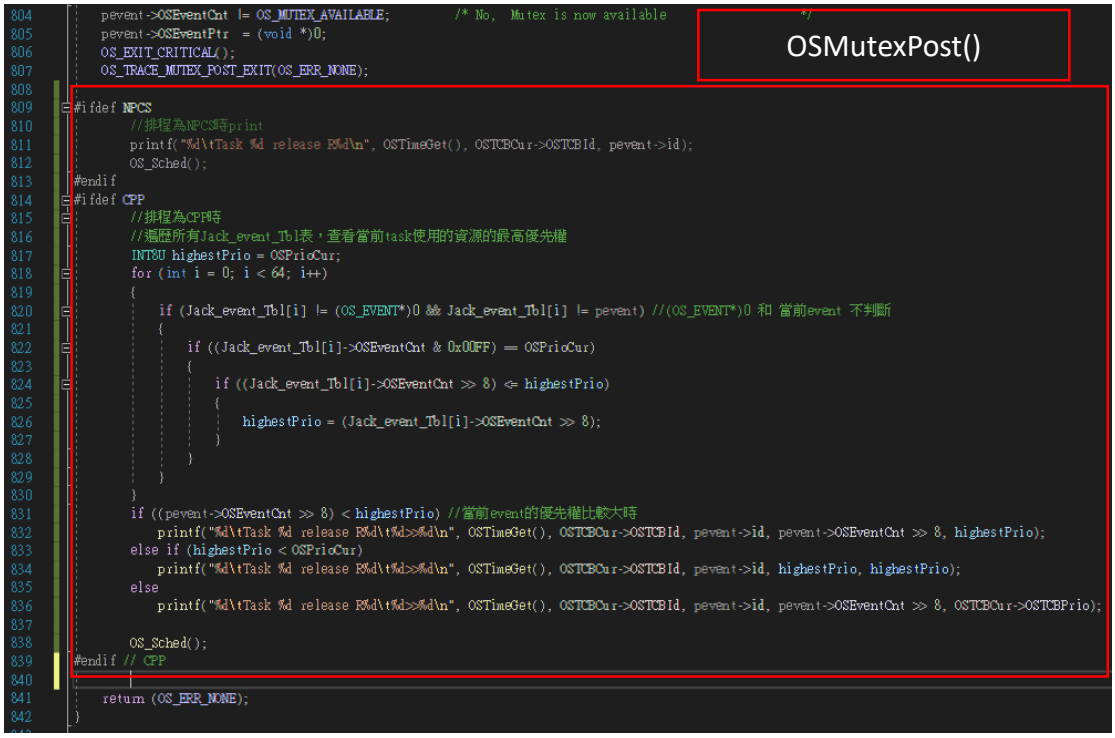
```
    pcp = (INT8U)(pevent->OSEventCnt >> 8u);                   /* Get PCP from mutex                       */
                                                               /* Is Mutex available?                      */
    if ((INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8) == OS_MUTEX_AVAILABLE) {
        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;
        }
        OS_TRACE_MUTEX_PEND_EXIT(*perr);

#ifdef NPCS
        //排程為NPCS時print
        printf("%d\tTask %d get R%d\n", OSTimeGet(), OSTCBCur->OSTCBId, pevent->id);
#endif

#ifdef CPP
        //排程為CPP時
        //遍歷所有Jack_event_Tbl表，查看當前task使用的資源的最高優先權
        INT8U highestPrio = OSPrioCur;
        for (int i = 0; i < 64; i++)
        {
            if (Jack_event_Tbl[i] != (OS_EVENT*)0 && Jack_event_Tbl[i] != pevent)  //(OS_EVENT*)0 和 當前event 不判斷
            {
                if ((Jack_event_Tbl[i]->OSEventCnt & 0x00FF) == OSPrioCur)
                {
                    if ((Jack_event_Tbl[i]->OSEventCnt >> 8) <= highestPrio)
                    {
                        highestPrio = (Jack_event_Tbl[i]->OSEventCnt >> 8);
                    }
                }
            }
        }
        if( (pevent->OSEventCnt >> 8) < highestPrio)  //當前event的優先權比較大時
            printf("%d\tTask %d get R%d\t\t%d>>%d\n", OSTimeGet(), OSTCBCur->OSTCBId, pevent->id, highestPrio, pevent->OSEventCnt >> 8);
        else if(highestPrio < OSPrioCur)
            printf("%d\tTask %d get R%d\t\t%d>>%d\n", OSTimeGet(), OSTCBCur->OSTCBId, pevent->id, highestPrio, highestPrio);
        else
            printf("%d\tTask %d get R%d\t\t%d>>%d\n", OSTimeGet(), OSTCBCur->OSTCBId, pevent->id, OSTCBCur->OSTCBPrio, pevent->OSEventCnt >> 8);
#endif // CPP

        return;
    }
```

OSMutexPend()




```
804     pevent->OSEventCnt |= OS_MUTEX_AVAILABLE;        /* No,  Mutex is now available                */
805     pevent->OSEventPtr  = (void *)0;
806     OS_EXIT_CRITICAL();
807     OS_TRACE_MUTEX_POST_EXIT(OS_ERR_NONE);
808
809 #ifdef NPCS
810         //排程為NPCS時print
811         printf("%d\tTask %d release R%d\n", OSTimeGet(), OSTCBCur->OSTCBId, pevent->id);
812         OS_Sched();
813 #endif
814 #ifdef CPP
815         //排程為CPP時
816         //遍歷所有Jack_event_Tbl表，查看當前task使用的資源的最高優先權
817         INT8U highestPrio = OSPrioCur;
818         for (int i = 0; i < 64; i++)
819         {
820             if (Jack_event_Tbl[i] != (OS_EVENT*)0 && Jack_event_Tbl[i] != pevent) //(OS_EVENT*)0 和 當前event 不判斷
821             {
822                 if ((Jack_event_Tbl[i]->OSEventCnt & 0x00FF) == OSPrioCur)
823                 {
824                     if ((Jack_event_Tbl[i]->OSEventCnt >> 8) <= highestPrio)
825                     {
826                         highestPrio = (Jack_event_Tbl[i]->OSEventCnt >> 8);
827                     }
828                 }
829             }
830         }
831         if ((pevent->OSEventCnt >> 8) < highestPrio) //當前event的優先權比較大時
832             printf("%d\tTask %d release R%d\t%d>>%d\n", OSTimeGet(), OSTCBCur->OSTCBId, pevent->id, pevent->OSEventCnt >> 8, highestPrio);
833         else if (highestPrio < OSPrioCur)
834             printf("%d\tTask %d release R%d\t%d>>%d\n", OSTimeGet(), OSTCBCur->OSTCBId, pevent->id, highestPrio, highestPrio);
835         else
836             printf("%d\tTask %d release R%d\t%d>>%d\n", OSTimeGet(), OSTCBCur->OSTCBId, pevent->id, pevent->OSEventCnt >> 8, OSTCBCur->OSTCBPrio);
837
838         OS_Sched();
839 #endif // CPP
840
841     return (OS_ERR_NONE);
842 }
```

[PART III] Performance Analysis [50%]

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

  [PART1]

  NPCS 跟 CPP 都是根據 RM 的方式，CPP 因為資源都只有自己的 task 在用所以排程的結果就是單純以 RM 排程的結果，而 NPCS 不同的是當 task 使用到資源時直到釋放掉才可以切換到其他的 task。

  [PART2]

  比較看結果 NPCS 與 CPP 的結果相同，因為 CPP 兩個 task 都有用到兩個資源所以當有 task 在使用資源時就不會被其他 task 中斷，結果就如同 NPCS 一樣只要在使用資源時直到釋放掉資源才可以切換到其他 task。

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

  NPCS 因為當有 task 在使用到資源時就會等到他做完才能切換到其他 task，所以不會有其他 task 能夠同時想使用資源就不會發生 deadlock。

  CPP 因為當 task 在使用資源時會繼承會使用到這個資源的最高 task 高一點點的優先權，只有比這個資源優先權還高的 task 才能中斷，但這些中斷的 task 不會用到同個資源，而會使用到同個資源的 task 因為無法中斷所以不會搶到同個資源，所以不會發生 deadlock。