嵌入式作業系統實作

Embedded OS Implementation

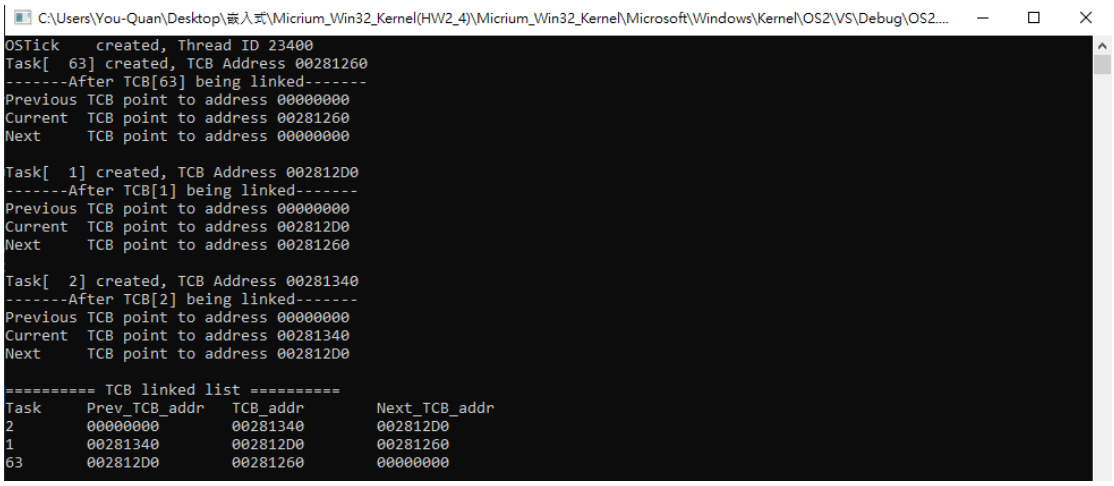
PA_1

指導教授: 陳雅淑 教授

課程學生: M10907314 張祐銓

[PART I] Task Control Block Linked List [20%]

- The screenshot results. (10%)



- A report that describes your implementation (please attach the screenshot of the code and MARK the modified part). (10%)

1. 上半部顯示 task 被創建時的 link，這段程式碼加在 OS_TCBInit 下，因為在執行 OSTaskCreateExt 建立 Task 內會執行到 OS_TCBInit 這個 function，OS_TCBInit 會初始化 task 的 OS_TCB 的數值，所以可以在這裡抓到 task 被建立時的 link 數值。

```
2241
2242        //OS_TCBInit會初始化task的OS_TCB所需要的變數
2243        //所以在這個地方print作業所需要的數值
2244        if (ptcb->OSTCBId == OS_TASK_IDLE_ID) //假如是idle_task的話就顯示優先權
2245        {
2246            printf("Task[  %d] created, TCB Address %p\n", ptcb->OSTCBPrio, ptcb);
2247            printf("-------After TCB[%d] being linked-------\n", ptcb->OSTCBPrio);
2248            printf("Previous TCB point to address %p\n", ptcb->OSTCBPrev);
2249            printf("Current\t TCB point to address %p\n", ptcb);
2250            printf("Next\t TCB point to address %p\n", ptcb->OSTCBNext);
2251            printf("\n");
2252        }
2253        else //反之顯示task的id
2254        {
2255            printf("Task[  %d] created, TCB Address %p\n", ptcb->OSTCBId, ptcb);
2256            printf("-------After TCB[%d] being linked-------\n", ptcb->OSTCBId);
2257            printf("Previous TCB point to address %p\n", ptcb->OSTCBPrev);
2258            printf("Current\t TCB point to address %p\n", ptcb);
2259            printf("Next\t TCB point to address %p\n", ptcb->OSTCBNext);
2260            printf("\n");
2261        }
```

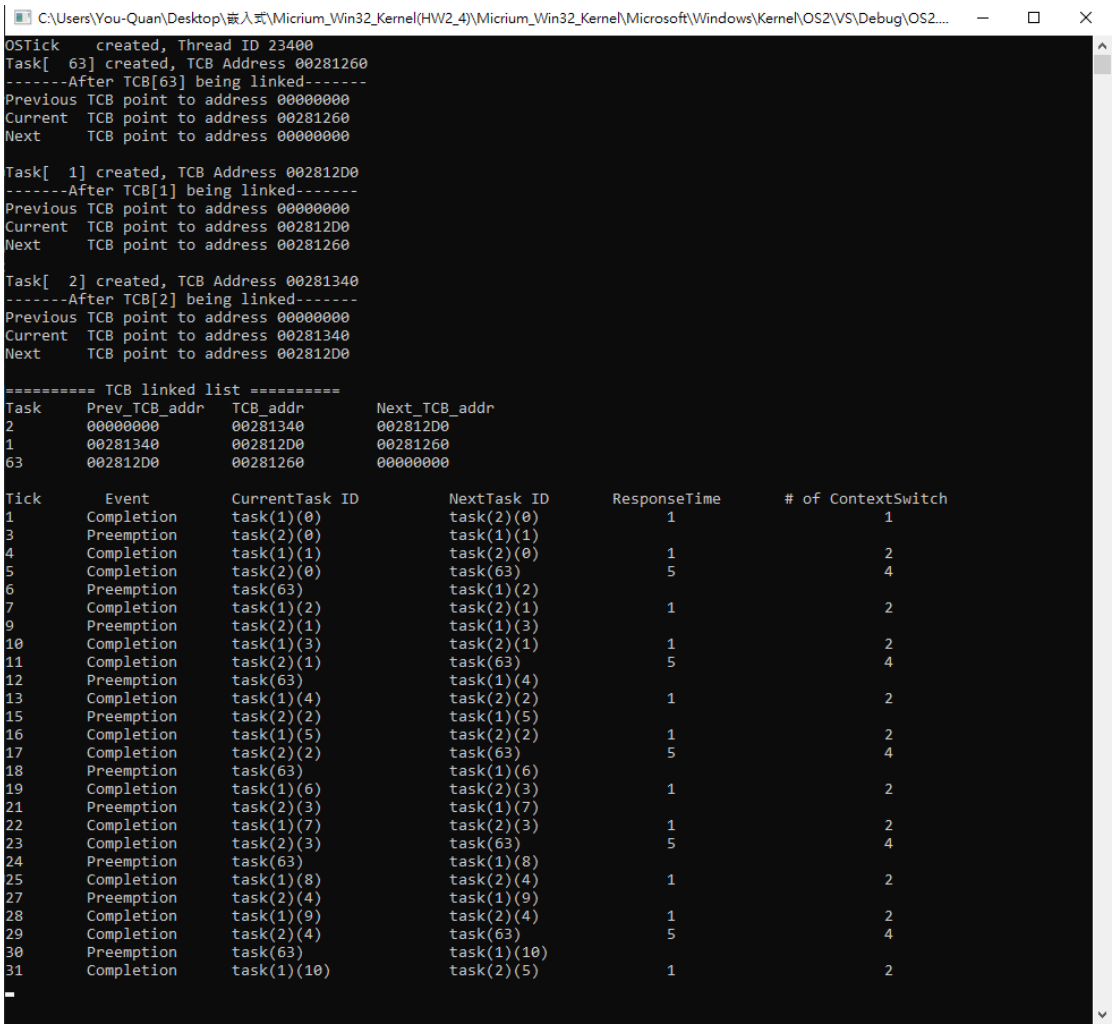
2. 下半部顯示 linked list 的表，取 OSTCBList 做開頭，使用一個 while 一直指向下一個 link 的 OS_TCB，然後顯示出所有 task 的 link 情況。

```
888  void  OSStart (void)
889  {
890      if (OSRunning == OS_FALSE) {
891
892          //----------linked list print------------
893          printf("========== TCB linked list ==========\n");
894          printf("Task\t Prev_TCB_addr\t TCB_addr\t Next_TCB_addr\n");
895
896          //作業要print所有task的linked list
897          //mytcb先取OSTCBList的OS_TCB
898          //之後mytcb = mytcb->OSTCBNext;一直等於下一個link的OS_TCB
899          //就能print出所有的task的link list
900          OS_TCB* mytcb = OSTCBList;
901          while (mytcb != (OS_TCB*)0)
902          {
903              if(mytcb->OSTCBId == OS_TASK_IDLE_ID)
904                  printf("%d\t %p\t %p\t %p\t\n", mytcb->OSTCBPrio, mytcb->OSTCBPrev, mytcb, mytcb->OSTCBNext);
905              else
906                  printf("%d\t %p\t %p\t %p\t\n", mytcb->OSTCBId, mytcb->OSTCBPrev, mytcb, mytcb->OSTCBNext);
907              mytcb = mytcb->OSTCBNext;
908          }
909          //----------------------------------------
910          printf("\nTick       Event         CurrentTask ID          NextTask ID       ResponseTime       # of ContextSwitch\n");
911
912          OS_SchedNew();                               /* Find highest priority's task priority number   */
913          OSPrioCur     = OSPrioHighRdy;
914          OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy]; /* Point to highest priority task ready to run    */
915          OSTCBCur      = OSTCBHighRdy;
916          OSStartHighRdy();                            /* Execute target specific code to start task     */
917      }
918  }
919
```

[PART II] RM Scheduler Implementation [80%]

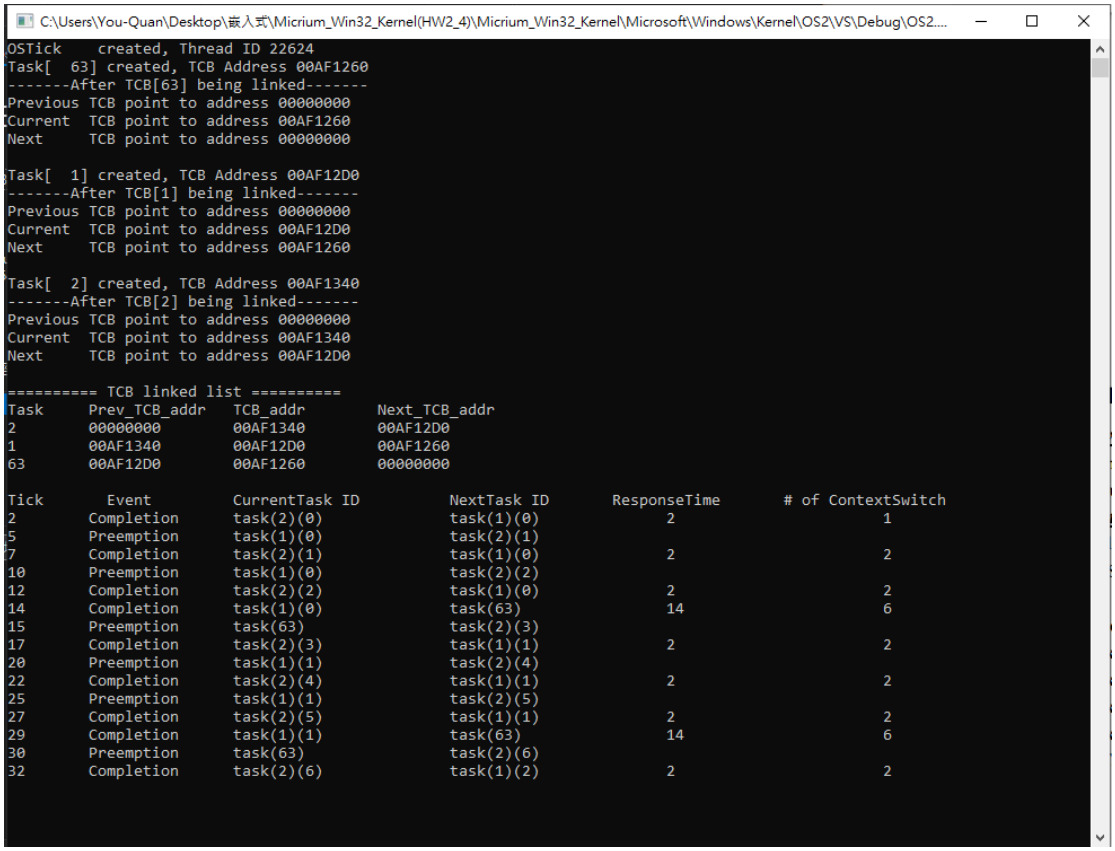
- The screenshot results (with the given format) of four task sets. (Time ticks 0-30 or miss deadline). (40%)

Task set 1 = {τ1 (0, 1, 3), τ2 (0, 3, 6)}

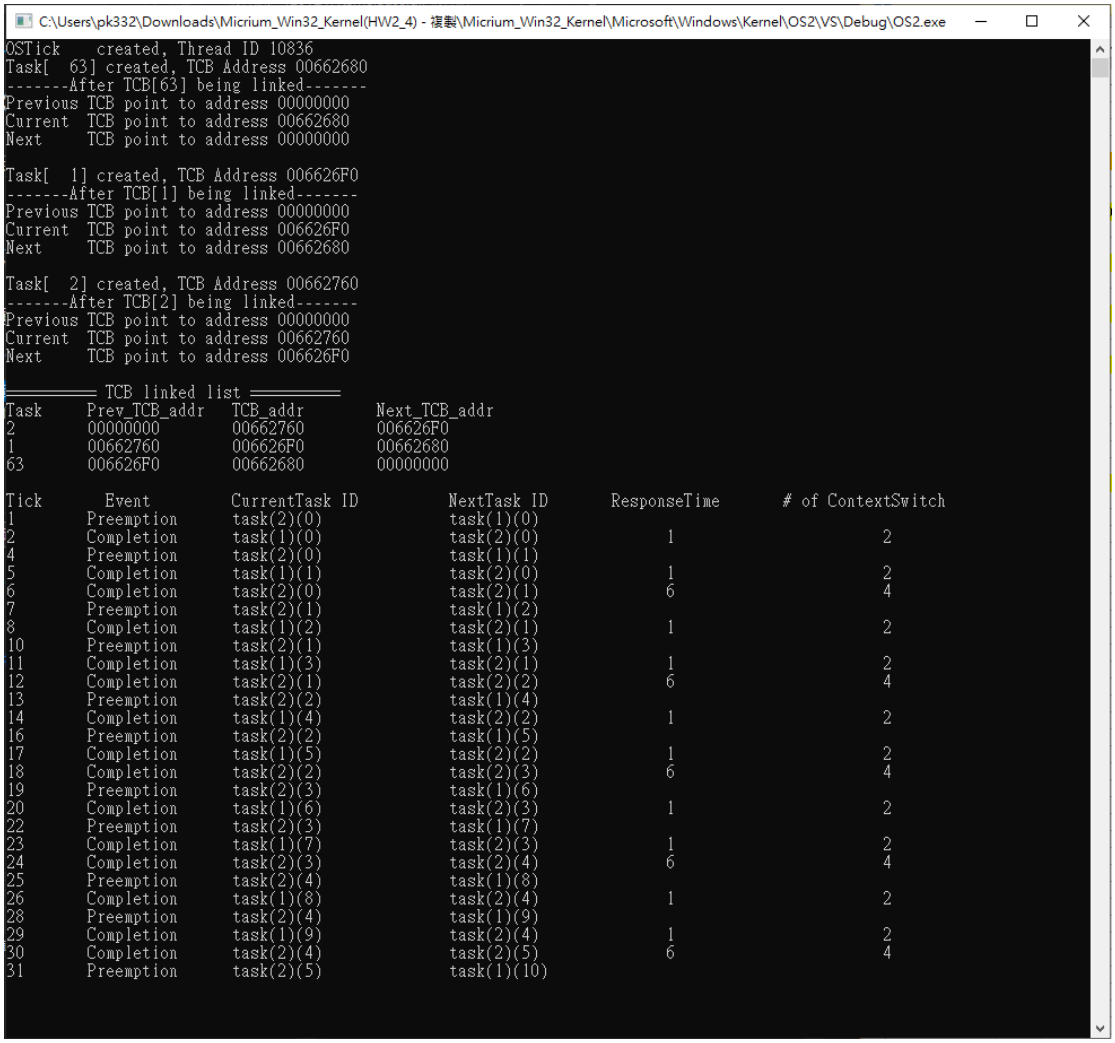


```
C:\Users\You-Quan\Desktop\嵌入式\Micrium_Win32_Kernel(HW2_4)\Micrium_Win32_Kernel\Microsoft\Windows\Kernel\OS2\VS\Debug\OS2....
OSTick    created, Thread ID 23400
Task[  63] created, TCB Address 00281260
-------After TCB[63] being linked-------
Previous TCB point to address 00000000
Current  TCB point to address 00281260
Next     TCB point to address 00000000

Task[  1] created, TCB Address 002812D0
-------After TCB[1] being linked-------
Previous TCB point to address 00000000
Current  TCB point to address 002812D0
Next     TCB point to address 00281260

Task[  2] created, TCB Address 00281340
-------After TCB[2] being linked-------
Previous TCB point to address 00000000
Current  TCB point to address 00281340
Next     TCB point to address 002812D0

========== TCB linked list ==========
Task     Prev_TCB_addr   TCB_addr        Next_TCB_addr
2        00000000        00281340        002812D0
1        00281340        002812D0        00281260
63       002812D0        00281260        00000000

Tick       Event         CurrentTask ID          NextTask ID       ResponseTime       # of ContextSwitch
1        Completion      task(1)(0)              task(2)(0)              1                       1
3        Preemption      task(2)(0)              task(1)(1)
4        Completion      task(1)(1)              task(2)(0)              1                       2
5        Completion      task(2)(0)              task(63)                5                       4
6        Preemption      task(63)                task(1)(2)
7        Completion      task(1)(2)              task(2)(1)              1                       2
9        Preemption      task(2)(1)              task(1)(3)
10       Completion      task(1)(3)              task(2)(1)              1                       2
11       Completion      task(2)(1)              task(63)                5                       4
12       Preemption      task(63)                task(1)(4)
13       Completion      task(1)(4)              task(2)(2)              1                       2
15       Preemption      task(2)(2)              task(1)(5)
16       Completion      task(1)(5)              task(2)(2)              1                       2
17       Completion      task(2)(2)              task(63)                5                       4
18       Preemption      task(63)                task(1)(6)
19       Completion      task(1)(6)              task(2)(3)              1                       2
21       Preemption      task(2)(3)              task(1)(7)
22       Completion      task(1)(7)              task(2)(3)              1                       2
23       Completion      task(2)(3)              task(63)                5                       4
24       Preemption      task(63)                task(1)(8)
25       Completion      task(1)(8)              task(2)(4)              1                       2
27       Preemption      task(2)(4)              task(1)(9)
28       Completion      task(1)(9)              task(2)(4)              1                       2
29       Completion      task(2)(4)              task(63)                5                       4
30       Preemption      task(63)                task(1)(10)
31       Completion      task(1)(10)             task(2)(5)              1                       2
```

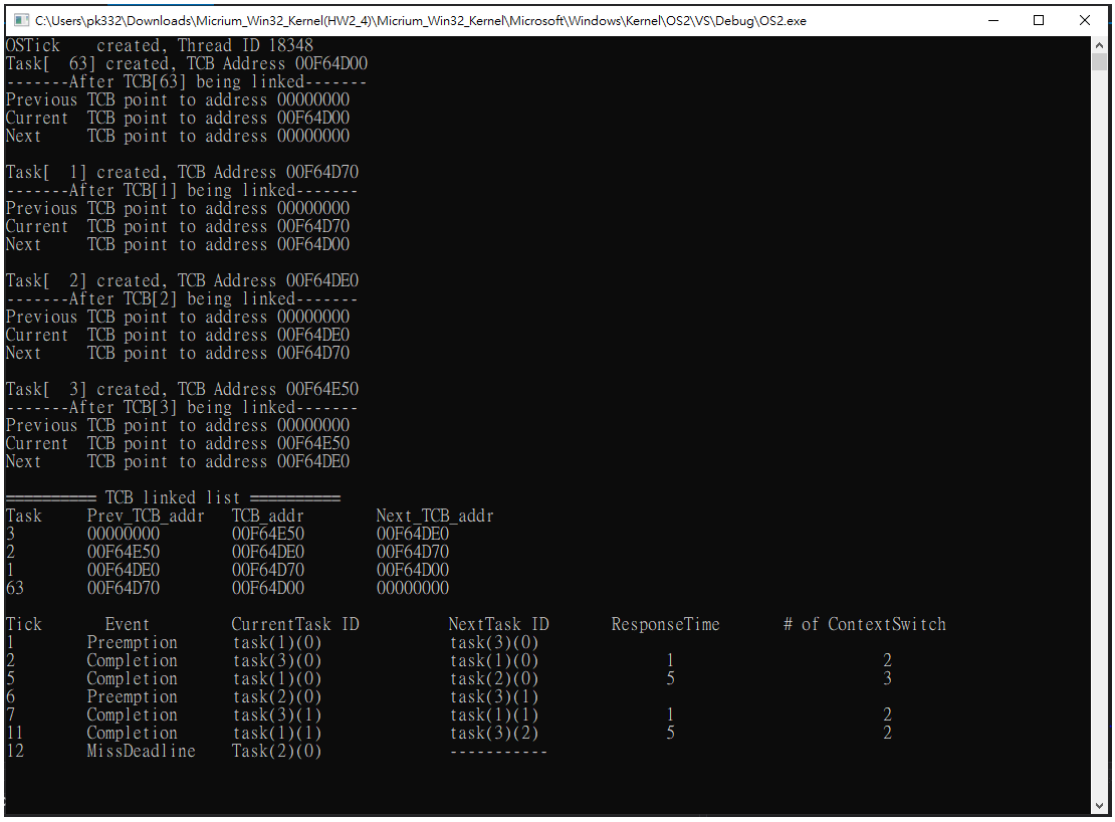
Task set 2 = {τ1 (0, 8, 15), τ2 (0, 2, 5)}



Task set 3 = {τ1 (1, 1, 3), τ2 (0, 4, 6)}



Task set 4 = {τ1 (0, 4, 6), τ2 (2, 2, 10), τ3 (1, 1, 5)}



- A report that describes your implementation (please attach the screenshot of the code and MARK the modified part). (40%)

實現方式條列簡述說明:

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。

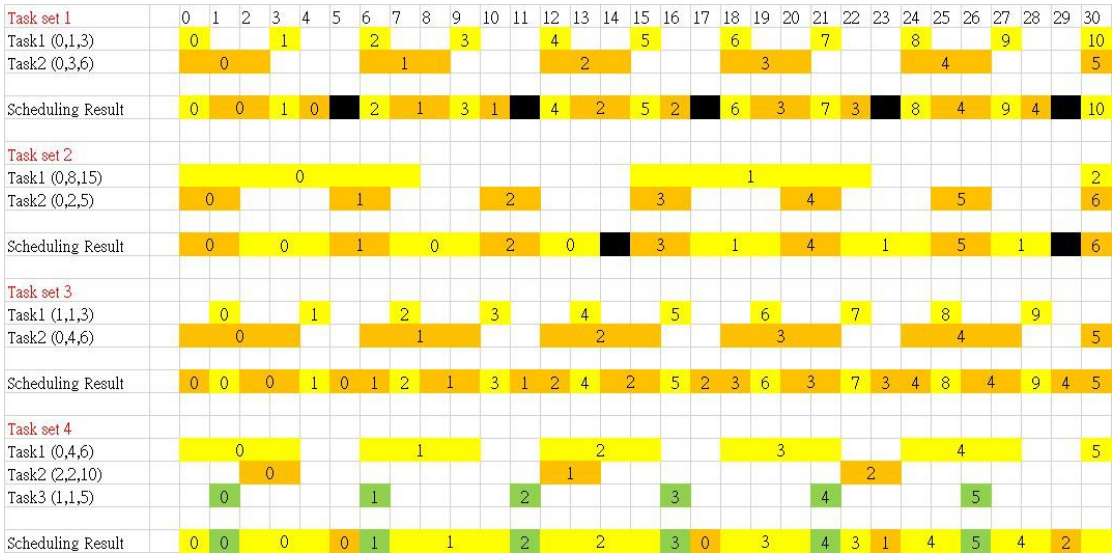
4. 當TimeTick中斷產生之後會進OSIntExit()把低優先權的task中斷給高優先權的task，所以在這裡顯示中斷狀態的字串。

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。

8. 以下是程式的截圖以及內部程式碼中較詳細的註解說明



Task set 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Task1 (0,1,3)	0			1			2			3			4			5			6			7			8			9			10
Task2 (0,3,6)	0						1						2						3						4						5
Scheduling Result	0	0	1	0		2	1	3	1		4	2	5	2		6	3	7	3		8	4	9	4		10					
Task set 2																															
Task1 (0,8,15)	0															1															2
Task2 (0,2,5)	0				1			2					3						4						5						6
Scheduling Result	0		0		1		0		2		0		3		1			4			1			5		1			6	
Task set 3																															
Task1 (1,1,3)		0			1			2			3			4			5			6			7			8			9		
Task2 (0,4,6)	0						1						2						3						4						5
Scheduling Result	0	0	0	1	0	1	2	1	3	1	2	4	2	5	2	3	6	3	7	3	4	8	4	9	4	5					
Task set 4																															
Task1 (0,4,6)		0					1						2						3						4						5
Task2 (2,2,10)			0										1										2								
Task3 (1,1,5)		0					1					2					3					4				5					
Scheduling Result	0	0		0		0	1		1		2		2			3	0		3		4	3	1	4		5	4		2		

Main.c

```
50    static void task1(void* p_arg);
51    static void task2(void* p_arg);
52    static void task3(void* p_arg);
53
54  /*
55    *********************************************************************************************************
56    *                                            LOCAL GLOBAL VARIABLES
57    *********************************************************************************************************
58    */
59
60    static  OS_STK  StartupTaskStk[APP_CFG_STARTUP_TASK_STK_SIZE];
61
62    #define TASK_STACKSIZE 2048
63
64    static OS_STK Task1_STK[TASK_STACKSIZE];
65    static OS_STK Task2_STK[TASK_STACKSIZE];
66    static OS_STK Task3_STK[TASK_STACKSIZE];
67
68    #define TASK1_ID 1 //定義task1的id、priority、arrival、execution、period
69    #define TASK1_PRIORITY 2
70    #define TASK1_ARRIVAL 0
71    #define TASK1_EXECUTION 4
72    #define TASK1_PERIOD 6
73
74    #define TASK2_ID 2 //定義task2的id、priority、arrival、execution、period
75    #define TASK2_PRIORITY 3
76    #define TASK2_ARRIVAL 2
77    #define TASK2_EXECUTION 2
78    #define TASK2_PERIOD 10
79
80    #define TASK3_ID 3 //定義task3的id、priority、arrival、execution、period
81    #define TASK3_PRIORITY 1
82    #define TASK3_ARRIVAL 1
83    #define TASK3_EXECUTION 1
84    #define TASK3_PERIOD 5
85
```

```
141 #ifdef TASK1_ID //假如task1有被定義的話 就創建task1工作
142     OSTaskCreateExt(task1,
143         0,
144         &Task1_STK[TASK_STACKSIZE - 1u],
145         TASK1_PRIORITY,
146         TASK1_ID,
147         &Task1_STK[0u],
148         TASK_STACKSIZE,
149         0u,
150         (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR),
151         TASK1_ARRIVAL,
152         TASK1_EXECUTION,
153         TASK1_PERIOD);
154 #endif // TASK1_ID
155
156
157 #ifdef TASK2_ID //假如task2有被定義的話 就創建task2工作
158     OSTaskCreateExt(task2,
159         0,
160         &Task2_STK[TASK_STACKSIZE - 1u],
161         TASK2_PRIORITY,
162         TASK2_ID,
163         &Task2_STK[0u],
164         TASK_STACKSIZE,
165         0u,
166         (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR),
167         TASK2_ARRIVAL,
168         TASK2_EXECUTION,
169         TASK2_PERIOD);
170 #endif // TASK2_ID
171
172 #ifdef TASK3_ID //假如task3有被定義的話 就創建task3工作
173     OSTaskCreateExt(task3,
174         0,
175         &Task3_STK[TASK_STACKSIZE - 1u],
176         TASK3_PRIORITY,
177         TASK3_ID,
178         &Task3_STK[0u],
179         TASK_STACKSIZE,
180         0u,
181         (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR),
182         TASK3_ARRIVAL,
183         TASK3_EXECUTION,
184         TASK3_PERIOD);
185 #endif // TASK3_ID
186
187     OSTimeSet(0); //重新歸零timetick
188
```

```
void task1(void* p_arg) //task1的執行程式
{
    (void)p_arg;

    while (1)
    {
        //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
        if (OSTCBCur->arrival > OSTimeGet())
        {
            OSTimeDly(OSTCBCur->arrival - OSTimeGet());
        }
        else
        {
            while (1) {

                //begin_ready_time是 task變為ready準備能執行的時間點
                //現在時間點 - task變為ready的時間點 < task的反應時間的話 就持續卡在while內
                while (OSTimeGet() - OSTCBCur->begin_ready_time < OSTCBCur->response) {
                    //Do something
                }

                //休息剩餘的時間(週期時間-反應時間)
                OSTimeDly(OSTCBCur->period - OSTCBCur->response);
            }
        }
    }
}
```

```
void task2(void* p_arg) //task2的執行程式
{
    (void)p_arg;

    while (1)
    {
        //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
        if (OSTCBCur->arrival > OSTimeGet())
        {
            OSTimeDly(OSTCBCur->arrival - OSTimeGet());
        }
        else
        {
            while (1) {

                //begin_ready_time是 task變為ready準備能執行的時間點
                //現在時間點 - task變為ready的時間點 < task的反應時間的話 就持續卡在while內
                while (OSTimeGet() - OSTCBCur->begin_ready_time < OSTCBCur->response) {
                    //Do something
                }

                //休息剩餘的時間(週期時間-反應時間)
                OSTimeDly(OSTCBCur->period - OSTCBCur->response);
            }
        }
    }
}
```

```
void task2(void* p_arg)
{
    (void)p_arg;

    while (1)
    {
        if (OSTCBCur->arrival > OSTimeGet())
        {
            OSTimeDly(OSTCBCur->arrival - OSTimeGet());
        }
        else
        {
            while (1) {

                while (OSTimeGet() - OSTCBCur->begin_ready_time < OSTCBCur->response) {
                    //Do something
                }
                OSTimeDly(OSTCBCur->period - OSTCBCur->response);
            }
        }
    }
}
```

```
void task3(void* p_arg) //task3的執行程式
{
    (void)p_arg;

    while (1)
    {
        //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
        if (OSTCBCur->arrival > OSTimeGet())
        {
            OSTimeDly(OSTCBCur->arrival - OSTimeGet());
        }
        else
        {
            while (1) {

                //begin_ready_time是 task變為ready準備能執行的時間點
                //現在時間點 - task變為ready的時間點 < task的反應時間的話 就持續卡在while內
                while (OSTimeGet() - OSTCBCur->begin_ready_time < OSTCBCur->response) {
                    //Do something
                }

                //休息剩餘的時間(週期時間-反應時間)
                OSTimeDly(OSTCBCur->period - OSTCBCur->response);
            }
        }
    }
}
```

ucos_ii.h

```
636
637     //在OS_TCB內新增變數
638
639     INT32U begin_ready_time; //紀錄task變ready的時間點
640     INT32U response; //反應時間
641     INT32U arrival; //到達時間
642     INT32U execution; //執行時間
643     INT32U period; //週期
644     INT32U job_id; //工作次數
645 } OS_TCB;
646
```

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

```
2106    INT8U  OS_TCBInit (INT8U    prio,
2107                       OS_STK  *ptos,
2108                       OS_STK  *pbos,
2109                       INT16U   id,
2110                       INT32U   stk_size,
2111                       void    *pext,
2112                       INT16U   opt,
2113                       INT32U   arrival,
2114                       INT32U   execution,
2115                       INT32U   period) //在OS_TCBInit內新增變數 arrival、exe
2116    {
2117        OS_TCB     *ptcb;
2118    #if OS_CRITICAL_METHOD == 3u                                    /* Allocate st
2119        OS_CPU_SR  cpu_sr = 0u;
2120    #endif
2121    #if OS_TASK_REG_TBL_SIZE > 0u
2122        INT8U      i;
2123    #endif
2124    #if OS_TASK_CREATE_EXT_EN > 0u
2125    #if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
2126        INT8U      j;
2127    #endif
2128    #endif
2129
2130
2131        OS_ENTER_CRITICAL();
2132        ptcb = OSTCBFreeList;                                       /* Get a free
2133        if (ptcb != (OS_TCB *)0) {
2134            OSTCBFreeList           = ptcb->OSTCBNext;              /* Update poin
2135            OS_EXIT_CRITICAL();
2136
2137            ptcb->begin_ready_time = arrival; //初始化在OS_TCB內新增的變數
2138            ptcb->response = execution;
2139            ptcb->arrival = arrival;
2140            ptcb->execution = execution;
2141            ptcb->period = period;
2142            ptcb->job_id = 0;
```

```
1028
1029        //這段是加在OSTimeTick內
1030        //從ReadyTable得到目前正在Ready的task
1031        //把現在正在Ready但卻沒有在執行的task的反應時間加一
1032        for (INT8U y = 0; y < 8; y++) //jack
1033        {
1034            for (INT8U x = 0; x < 8; x++)
1035            {
1036
1037                if ((OSRdyTbl[y] & (INT8U)1 << x) == (INT8U)1 << x) //確認是有有Ready
1038                {
1039                    INT8U prio = (y << 3u) + x;
1040                    if (OSTCBPrioTbl[prio] != OSTCBCur && prio != OS_TASK_IDLE_PRIO)
1041                    {
1042                        if (OSTimeGet() > OSTCBPrioTbl[prio]->arrival)
1043                        {
1044                            OSTCBPrioTbl[prio]->response += 1;
1045
1046                            //假設反應時間 大於週期 表示說task已經missdeadline了
1047                            if (OSTCBPrioTbl[prio]->response > OSTCBPrioTbl[prio]->period)
1048                            {
1049                                printf("%d\t MissDeadline\t Task(%d)(%d)\t\t ----------\n", OSTimeGet(),
1050                                    OSTCBPrioTbl[prio]->OSTCBId, OSTCBPrioTbl[prio]->job_id);
1051                                while (1) {}
1052                            }
1053                        }
1054                    }
1055                }
1056            }
1057        }
```

```
1054        ptcb = OSTCBList;                                      /* Point at first TCB in TCB list               */
1055        while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {         /* Go through all TCBs in TCB list              */
1056            OS_ENTER_CRITICAL();
1057            if (ptcb->OSTCBDly != 0u) {                        /* No, Delayed or waiting for event with TO     */
1058                ptcb->OSTCBDly--;                              /* Decrement nbr of ticks to end of delay       */
1059                if (ptcb->OSTCBDly == 0u) {                    /* Check for timeout                            */
1060
1061                    if ((ptcb->OSTCBStat & OS_STAT_PEND_ANY) != OS_STAT_RDY) {
1062                        ptcb->OSTCBStat  &= (INT8U)~(INT8U)OS_STAT_PEND_ANY;   /* Yes, Clear status flag   */
1063                        ptcb->OSTCBStatPend = OS_STAT_PEND_TO;                 /* Indicate PEND timeout    */
1064                    } else {
1065                        ptcb->OSTCBStatPend = OS_STAT_PEND_OK;
1066                    }
1067
1068                    if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == OS_STAT_RDY) {  /* Is task suspended?       */
1069                        OSRdyGrp               |= ptcb->OSTCBBitY;             /* No,  Make ready          */
1070                        OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
1071
1072                        //這段是加在OSTimeTick內
1073                        //當某個task轉變為Ready的時間 設定變數數值
1074                        ptcb->begin_ready_time = OSTimeGet();  //begin_ready_time是task轉變為Ready的時間點
1075                        ptcb->response = ptcb->execution; //反應時間等於執行時間
1076                        ptcb->OSTCBCtxSwCtr = 0; //context switch次數設為0
1077
1078                        OS_TRACE_TASK_READY(ptcb);
1079                    }
1080                }
1081            }
1082            ptcb = ptcb->OSTCBNext;                            /* Point at next TCB in TCB list                */
1083            OS_EXIT_CRITICAL();
1084        }
1085    }
```

```
1779  void  OS_Sched (void)
1780  {
1781  #if OS_CRITICAL_METHOD == 3u                       /* Allocate storage for CPU status register     */
1782      OS_CPU_SR  cpu_sr = 0u;
1783  #endif
1784
1785
1786      OS_ENTER_CRITICAL();
1787      if (OSIntNesting == 0u) {                      /* Schedule only if all ISRs done and ...       */
1788          if (OSLockNesting == 0u) {                 /* ... scheduler is not locked                  */
1789              OS_SchedNew();
1790              OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
1791              if (OSPrioHighRdy != OSPrioCur) {          /* No Ctx Sw if current task is highest rdy     */
1792
1793                  if (OSTimeGet() > OSTCBCur->arrival)
1794                  {
1795                      OSTCBCur->OSTCBCtxSwCtr++; //OSTCBCtxSwCtr加一
1796
1797                      //task完成之後會執行OS_Sched所以在這裡print完成
1798                      //假設是task完成後切入idle task，print的方式會不一樣
1799                      if(OSTCBHighRdy->OSTCBPrio == OS_TASK_IDLE_PRIO)
1800                          printf("%d\t Completion\t task(%d)(%d)\t\t task(%d)  \t\t %d\t\t\t\t %d\n", OSTimeGet(),
1801                              OSTCBCur->OSTCBId, OSTCBCur->job_id,
1802                              OS_TASK_IDLE_PRIO,
1803                              OSTimeGet() - OSTCBCur->begin_ready_time, OSTCBCur->OSTCBCtxSwCtr);
1804                      else
1805                          printf("%d\t Completion\t task(%d)(%d)\t\t task(%d)(%d)\t\t %d\t\t\t\t %d\n", OSTimeGet(),
1806                              OSTCBCur->OSTCBId, OSTCBCur->job_id,
1807                              OSTCBHighRdy->OSTCBId, OSTCBHighRdy->job_id,
1808                              OSTimeGet() - OSTCBCur->begin_ready_time, OSTCBCur->OSTCBCtxSwCtr);
1809
1810                      //task完成所以將變數初始化
1811                      OSTCBCur->begin_ready_time = OSTimeGet();
1812                      OSTCBCur->response = OSTCBCur->execution;
1813                      OSTCBCur->OSTCBCtxSwCtr = 0;
1814
1815                      //完成次數+1
1816                      OSTCBCur->job_id += 1;
1817
1818  #if OS_TASK_PROFILE_EN > 0u
1819                      OSTCBHighRdy->OSTCBCtxSwCtr++;         /* Inc. # of context switches to this task      */
```

```
695   void  OSIntExit (void)
696   {
697   #if OS_CRITICAL_METHOD == 3u                       /* Allocate storage for CPU status register */
698       OS_CPU_SR  cpu_sr = 0u;
699   #endif
700
701       if (OSRunning == OS_TRUE) {
702           OS_ENTER_CRITICAL();
703           if (OSIntNesting > 0u) {                       /* Prevent OSIntNesting from wrapping       */
704               OSIntNesting--;
705           }
706           if (OSIntNesting == 0u) {                      /* Reschedule only if all ISRs complete ... */
707               if (OSLockNesting == 0u) {                 /* ... and not locked.                      */
708
709                   //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
710                   //迴避掉這次切換讓原本的task先做完
711                   if (OSTimeGet() - OSTCBCur->begin_ready_time != OSTCBCur->response)
712                   {
713                       OS_SchedNew();
714                       OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
715                       if (OSPrioHighRdy != OSPrioCur) {          /* No Ctx Sw if current task is highest rdy */
716
717                           OSTCBCur->OSTCBCtxSwCtr++; //OSTCBCtxSwCtr加一
718
719                           //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
720                           //假設是idle task被搶占，print的方式會不一樣
721                           if (OSTCBCur->OSTCBPrio == OS_TASK_IDLE_PRIO)
722                               printf("%d\t Preemption\t task(%d)  \t\t task(%d)(%d)\n", OSTimeGet(),
723                                   OS_TASK_IDLE_PRIO,
724                                   OSTCBHighRdy->OSTCBId, OSTCBHighRdy->job_id);
725                           else
726                               printf("%d\t Preemption\t task(%d)(%d)\t\t task(%d)(%d)\n", OSTimeGet(),
727                                   OSTCBCur->OSTCBId, OSTCBCur->job_id,
728                                   OSTCBHighRdy->OSTCBId, OSTCBHighRdy->job_id);
729
730   #if OS_TASK_PROFILE_EN > 0u
731                           OSTCBHighRdy->OSTCBCtxSwCtr++;         /* Inc. # of context switches to this task  */
732   #endif
```

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
63
64      if (OSIntNesting > 0u) {                    /* See if trying to call from an ISR
65          return;
66      }
67      if (OSLockNesting > 0u) {                   /* See if called with scheduler locked
68          return;
69      }
70      if (ticks > 0u) {                           /* 0 means no delay!
71          OS_ENTER_CRITICAL();
72          y             =  OSTCBCur->OSTCBY;      /* Delay current task
73          OSRdyTbl[y] &= (OS_PRIO)~OSTCBCur->OSTCBBitX;
74          OS_TRACE_TASK_SUSPENDED(OSTCBCur);
75          if (OSRdyTbl[y] == 0u) {
76              OSRdyGrp &= (OS_PRIO)~OSTCBCur->OSTCBBitY;
77          }
78          OSTCBCur->OSTCBDly = ticks;             /* Load ticks in TCB
79          OS_TRACE_TASK_DLY(ticks);
80          OS_EXIT_CRITICAL();
81          OS_Sched();                             /* Find next task to run!
82      }
83
84      //當發生執行 OSTimeDly(0) 的情況表示反應時間剛好是週期時間
85      //task已經完成這次週期的工作要做下個週期的工作
86      if (OSTCBCur->period == OSTCBCur->response)
87      {
88          //print這次週期的工作完成，輪到下個週期的工作開始
89          printf("%d\t Completion\t task(%d)(%d)\t\t task(%d)(%d)\t\t %d\t\t\t\t %d\n", OSTimeGet(),
90              OSTCBCur->OSTCBId, OSTCBCur->job_id,
91              OSTCBCur->OSTCBId, OSTCBCur->job_id + 1,
92              OSTimeGet() - OSTCBCur->begin_ready_time, OSTCBCur->OSTCBCtxSwCtr);
93
94          //初始化task的OS_TCB變數參數
95          OSTCBCur->begin_ready_time = OSTimeGet();
96          OSTCBCur->response = OSTCBCur->execution;
97          OSTCBCur->OSTCBCtxSwCtr = 0;
98
99          //完成工作次數+1
100         OSTCBCur->job_id += 1;
101     }
102 }
```

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_cpu_c.c、main.c、os_time.c、ucos_ii.h、os_task.c、os_core.c