# 嵌入式作業系統實作

# Embedded OS Implementation

# PA_2

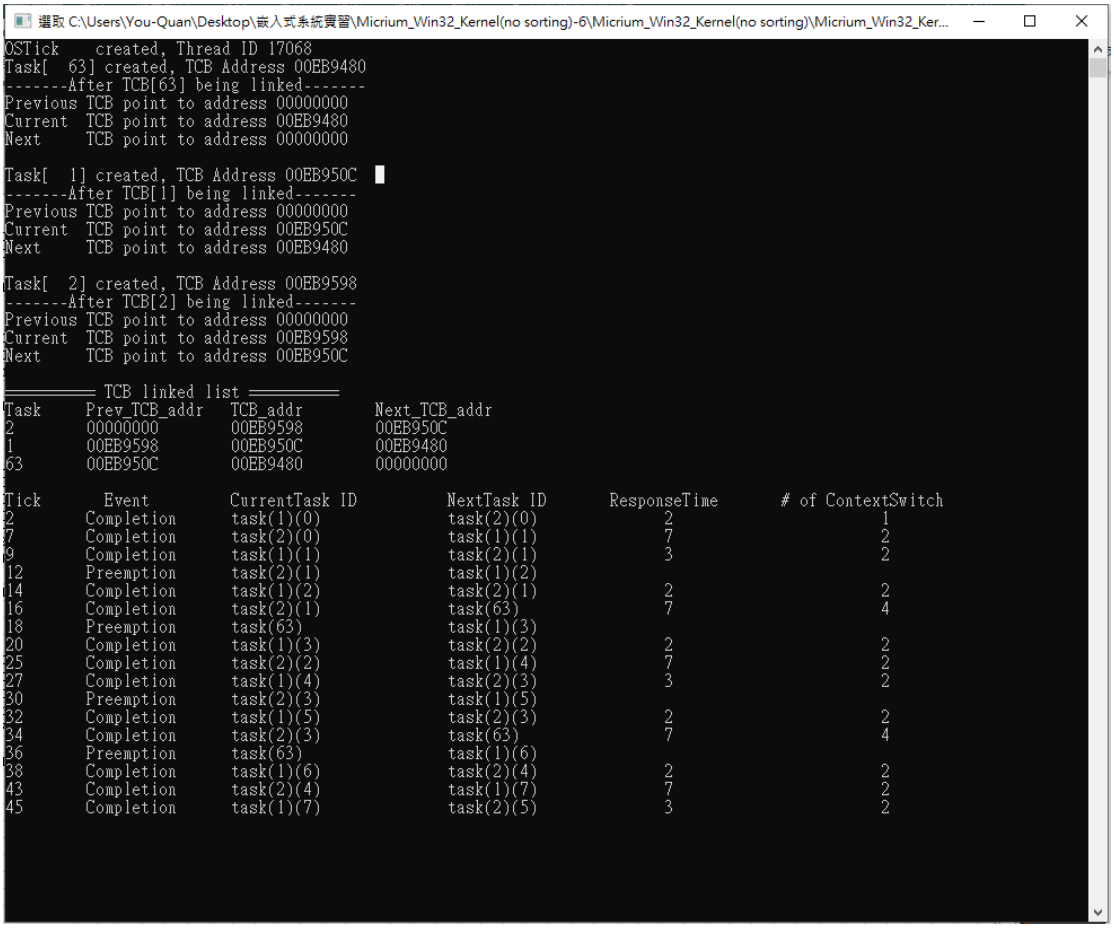
指導教授: 陳雅淑 教授

課程學生: M10907314 張祐銓

# [ PART I] EDF Scheduler Implementation [60%]

- The screenshot results (with the given format) of two task sets. (Tick 0 to tick 40 or the tick when a task missing the deadline) (10%)

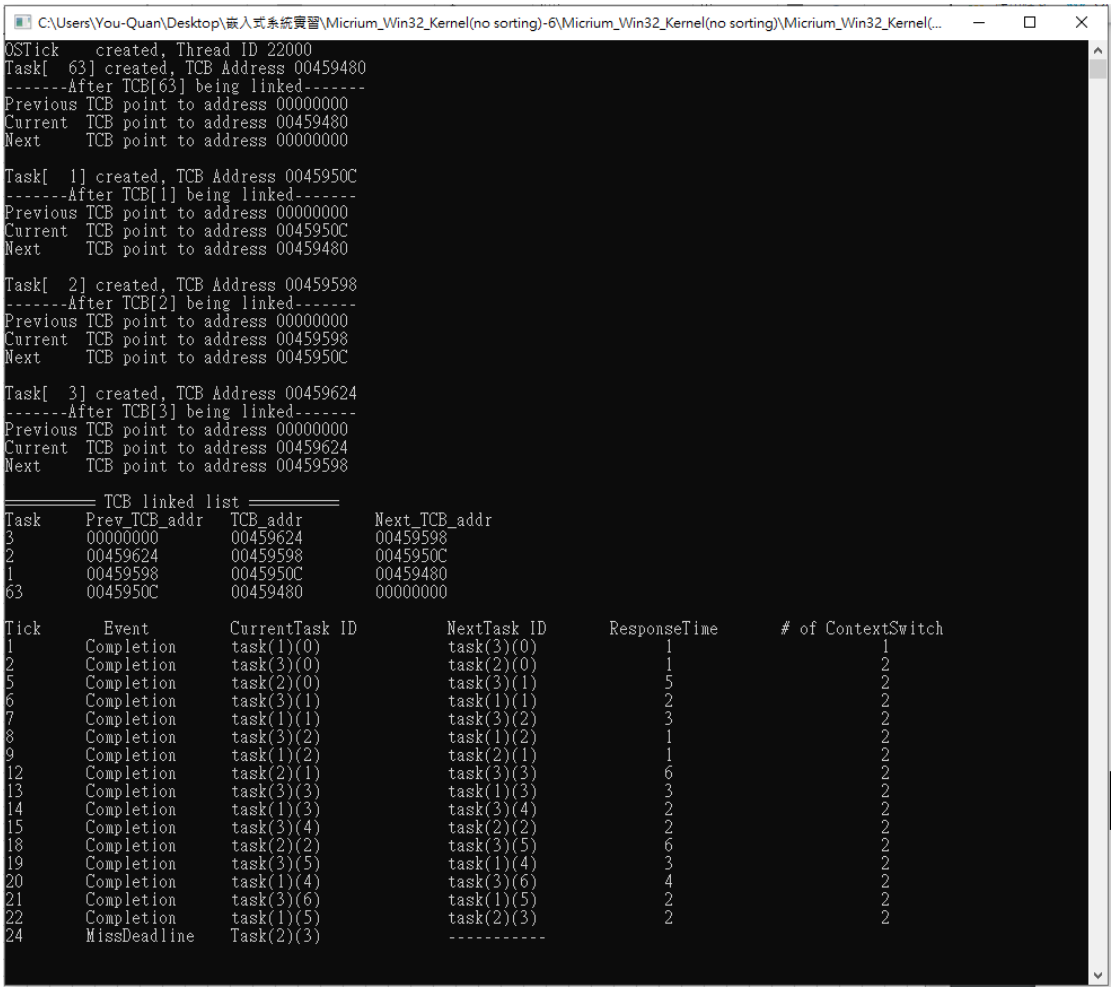
  Task set 1 = {τ1 (0, 2, 6), τ2 (0, 5, 9)}



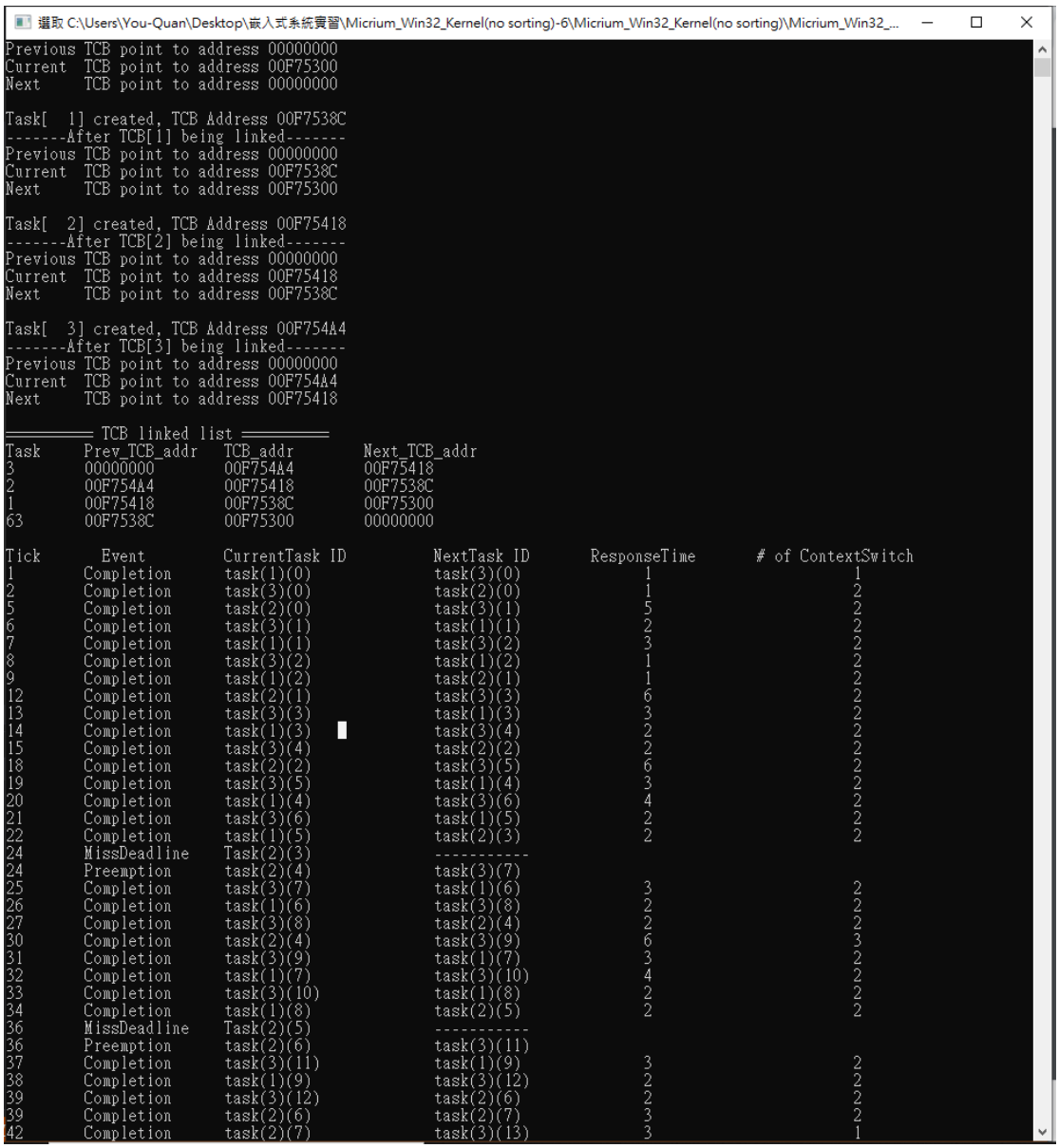
Task set 2 = {τ1 (0, 1, 4), τ2 (0, 3, 6), τ3 (1, 1, 3)}

做了兩個版本

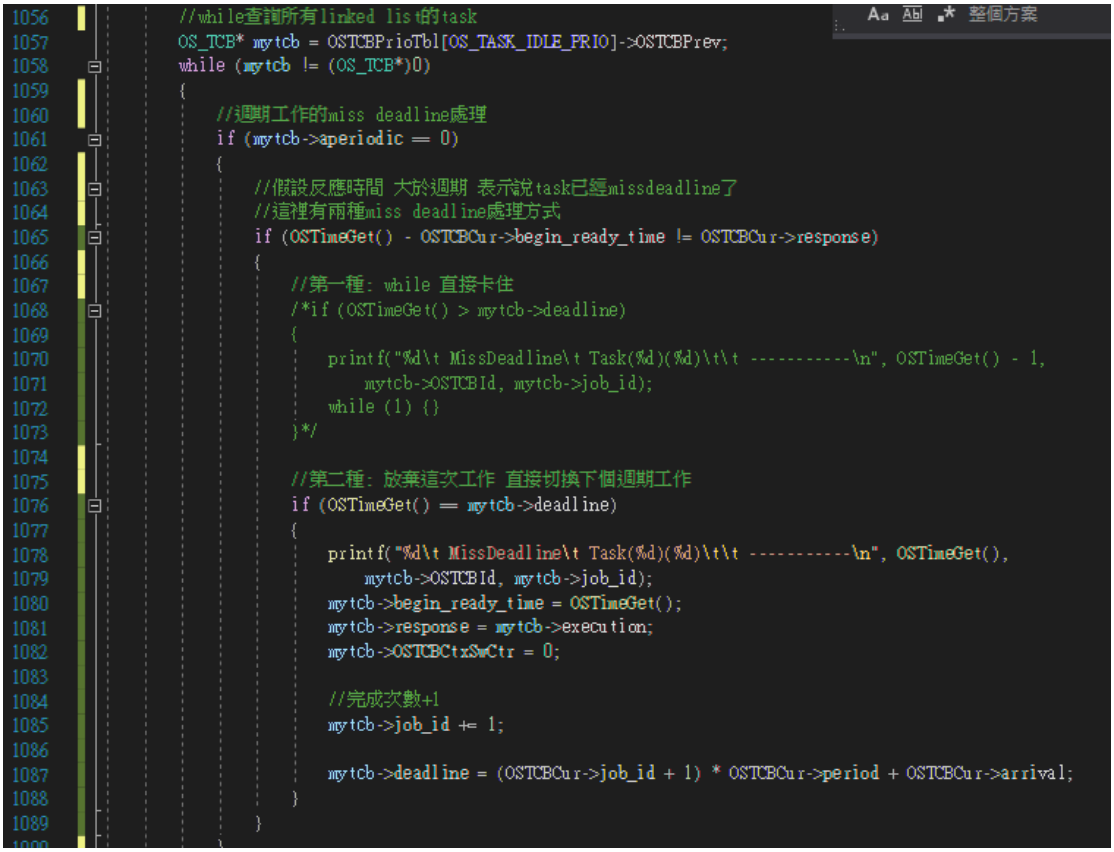
1. Missdeadline 就卡 while



2. 放棄掉這次的missdeadline task



- Implement and describe how to handle the deadline missing situation under EDF. (10%)

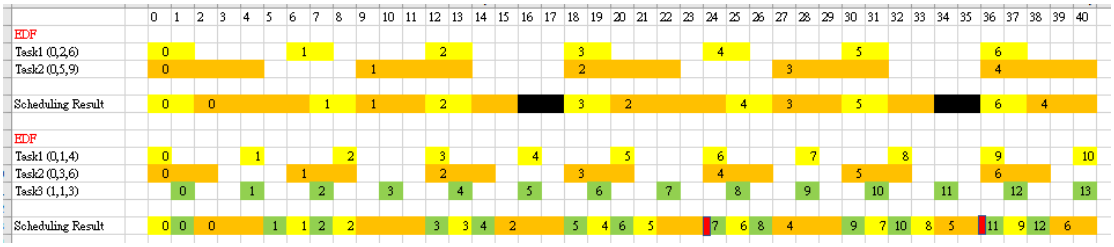


```
1056        //while查詢所有linked list的task
1057        OS_TCB* mytcb = OSTCBPrioTbl[OS_TASK_IDLE_PRIO]->OSTCBPrev;
1058        while (mytcb != (OS_TCB*)0)
1059        {
1060            //週期工作的miss deadline處理
1061            if (mytcb->aperiodic == 0)
1062            {
1063                //假設反應時間 大於週期 表示說task已經missdeadline了
1064                //這裡有兩種miss deadline處理方式
1065                if (OSTimeGet() - OSTCBCur->begin_ready_time != OSTCBCur->response)
1066                {
1067                    //第一種: while 直接卡住
1068                    /*if (OSTimeGet() > mytcb->deadline)
1069                    {
1070                        printf("%d\t MissDeadline\t Task(%d)(%d)\t\t ----------\n", OSTimeGet() - 1,
1071                            mytcb->OSTCBId, mytcb->job_id);
1072                        while (1) {}
1073                    }*/
1074
1075                    //第二種: 放棄這次工作 直接切換下個週期工作
1076                    if (OSTimeGet() == mytcb->deadline)
1077                    {
1078                        printf("%d\t MissDeadline\t Task(%d)(%d)\t\t ----------\n", OSTimeGet(),
1079                            mytcb->OSTCBId, mytcb->job_id);
1080                        mytcb->begin_ready_time = OSTimeGet();
1081                        mytcb->response = mytcb->execution;
1082                        mytcb->OSTCBCtxSwCtr = 0;
1083
1084                        //完成次數+1
1085                        mytcb->job_id += 1;
1086
1087                        mytcb->deadline = (OSTCBCur->job_id + 1) * OSTCBCur->period + OSTCBCur->arrival;
1088                    }
1089                }
```

判斷 miss deadline 的程式打在 OSTimeTick()下，只要 OSTimeGet() 大於某個 task 的 deadline 就判斷為 miss deadline，之後做了兩種處理 miss deadline 的版本。

1. 直接卡一個 while，讓整個系統停下來。
2. 放棄這次的 task 工作，將 OS_TCB 的變數設定為下個週期工作的數值，之後過 OSTimeTick()會進入 OSIntExit()進行 context switch，讓系統繼續運作下去。

- 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。

EDF 的實現條列說明:

1. 在OS_TCB中新增變數

   ```
   INT32U deadline; //EDF的最後期限
   ```

2. OS_SchedNew()會去找下個要執行的 task，將 EDF 找下個 task 的程式寫在 OS_SchedNew()中，根據所有 task 的 deadline 大小決定哪個 task 要優先執行，最後直接改動 OSPrioHighRdy 變成我們希望執行的 task。

3. 在 OSTimeDly(0)時為 task 剛好完成在 deadline 前，在 OSTimeDly(0)時執行 OS_Sched 做排程。

4. 在 OSIntExit()在 task 快要完成時，將 task 的 deadline 先切換到下個 deadline 時間，判斷下個要換哪個 task 執行，先切換是因為 task 快完成時會迴避掉這次的 context switch。

- 程式截圖部分跟 CUS 的程式截圖整合在一起

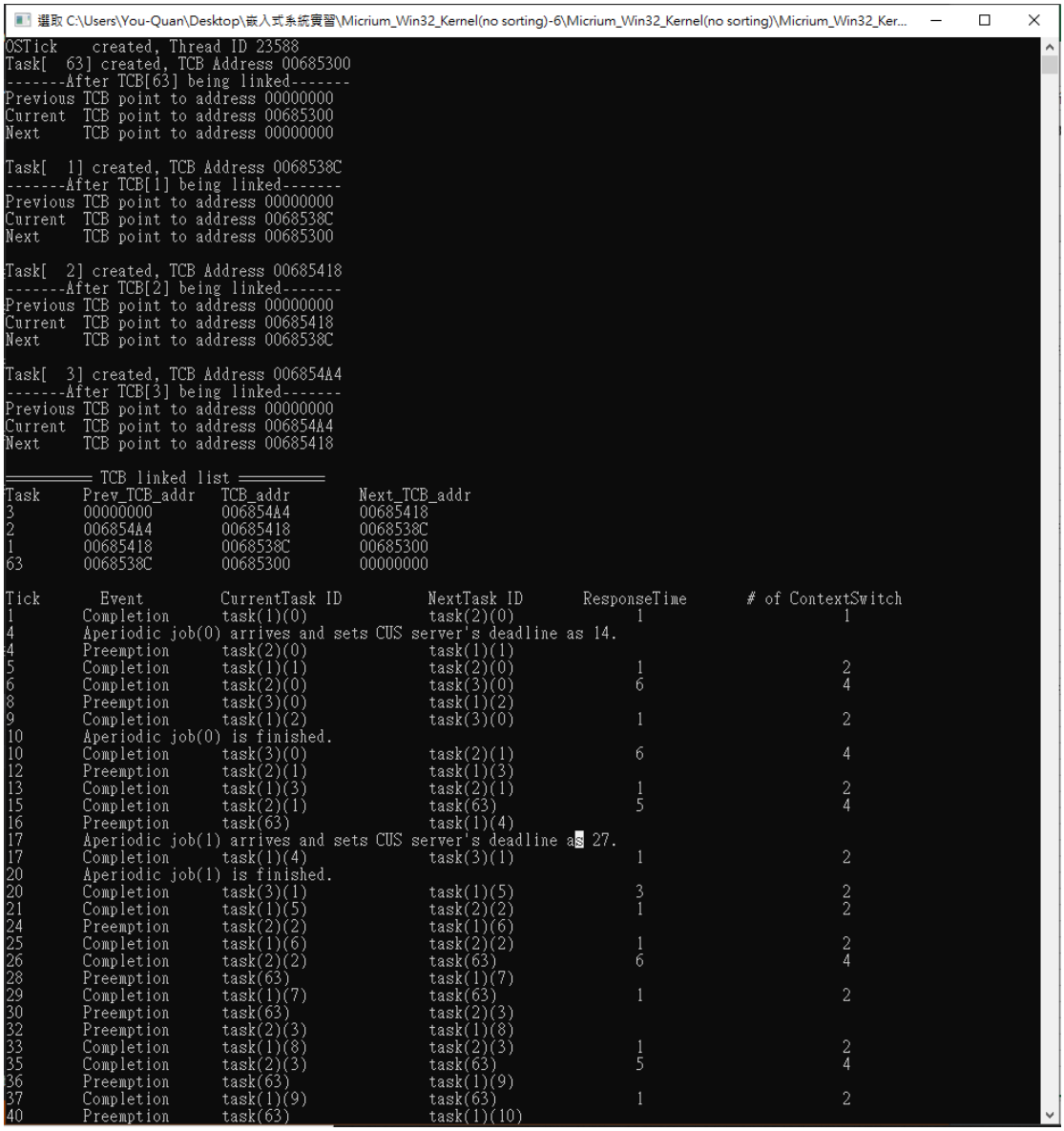
# [ PART II] CUS Scheduler Implementation [40%]

- The screenshot results (with the given format) of two task sets. (Tick 0 to tick 40 or the tick when a task missing the deadline). (10%)

========== **Task Set 1** ==========

**Periodic Task Set1 = {τ1 (0, 1, 4), τ2 (0, 4, 10), τ3_ServerSize (0.3)}**

**Aperiodic Jobs Set1 = {j0 (4, 3, 16), j1 (17, 3, 30)}**

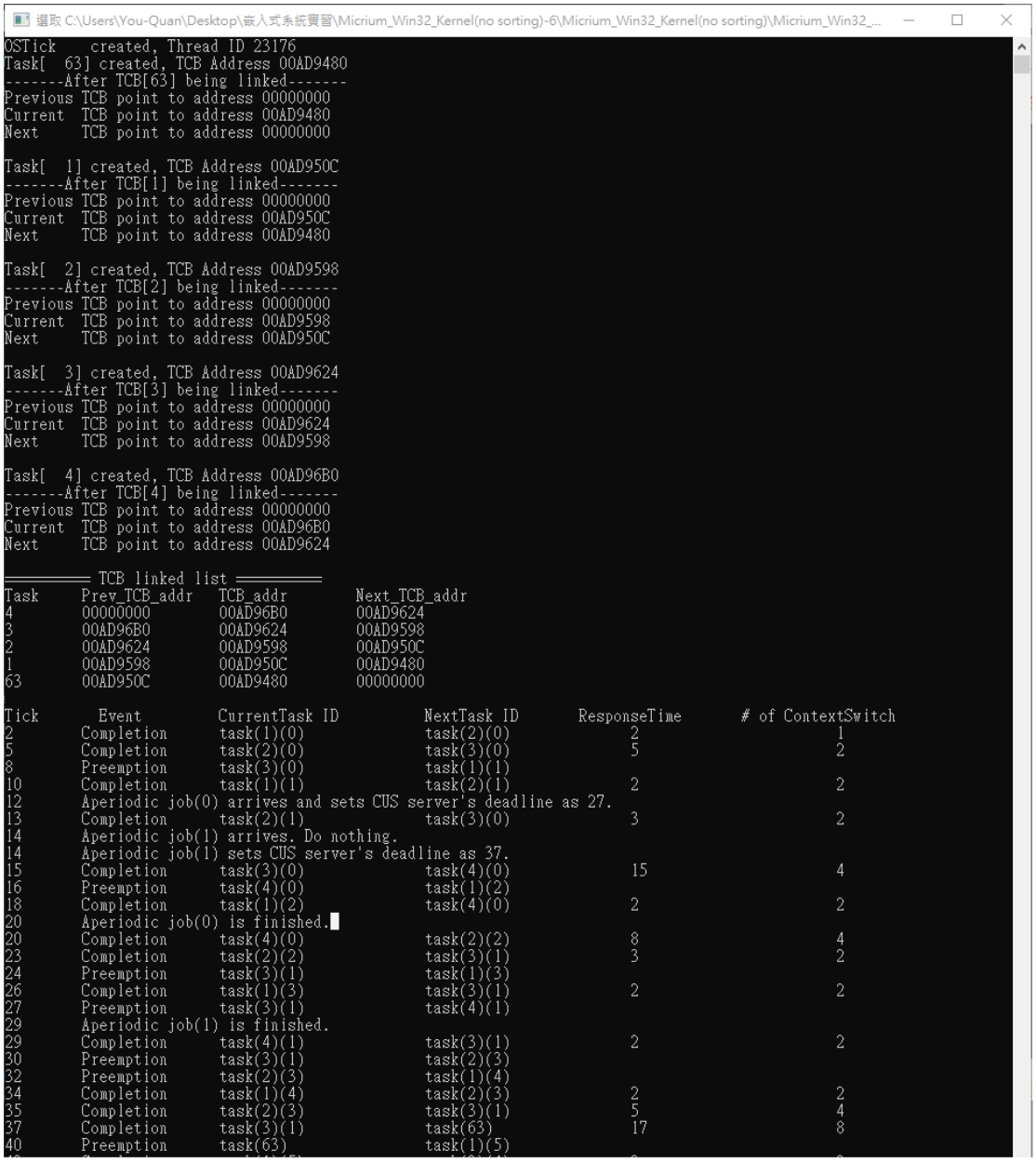


```
選取 C:\Users\You-Quan\Desktop\嵌入式系統實習\Micrium_Win32_Kernel(no sorting)-6\Micrium_Win32_Kernel(no sorting)\Micrium_Win32_Ker...
OSTick    created, Thread ID 23588
Task[  63] created, TCB Address 00685300
-------After TCB[63] being linked-------
Previous TCB point to address 00000000
Current  TCB point to address 00685300
Next     TCB point to address 00000000

Task[  1] created, TCB Address 0068538C
-------After TCB[1] being linked-------
Previous TCB point to address 00000000
Current  TCB point to address 0068538C
Next     TCB point to address 00685300

Task[  2] created, TCB Address 00685418
-------After TCB[2] being linked-------
Previous TCB point to address 00000000
Current  TCB point to address 00685418
Next     TCB point to address 0068538C

Task[  3] created, TCB Address 006854A4
-------After TCB[3] being linked-------
Previous TCB point to address 00000000
Current  TCB point to address 006854A4
Next     TCB point to address 00685418

========= TCB_linked list =========
Task     Prev_TCB_addr   TCB_addr        Next_TCB_addr
3        00000000        006854A4        00685418
2        006854A4        00685418        0068538C
1        00685418        0068538C        00685300
63       0068538C        00685300        00000000

Tick       Event         CurrentTask ID          NextTask ID       ResponseTime       # of ContextSwitch
1        Completion      task(1)(0)              task(2)(0)              1                       1
4        Aperiodic job(0) arrives and sets CUS server's deadline as 14.
4        Preemption      task(2)(0)              task(1)(1)
5        Completion      task(1)(1)              task(2)(0)              1                       2
6        Completion      task(2)(0)              task(3)(0)              6                       4
8        Preemption      task(3)(0)              task(1)(2)
9        Completion      task(1)(2)              task(3)(0)              1                       2
10       Aperiodic job(0) is finished.
10       Completion      task(3)(0)              task(2)(1)              6                       4
12       Preemption      task(2)(1)              task(1)(3)
13       Completion      task(1)(3)              task(2)(1)              1                       2
15       Completion      task(2)(1)              task(63)                5                       4
16       Preemption      task(63)                task(1)(4)
17       Aperiodic job(1) arrives and sets CUS server's deadline as 27.
17       Completion      task(1)(4)              task(3)(1)              1                       2
20       Aperiodic job(1) is finished.
20       Completion      task(3)(1)              task(1)(5)              3                       2
21       Completion      task(1)(5)              task(2)(2)              1                       2
24       Preemption      task(2)(2)              task(1)(6)
25       Completion      task(1)(6)              task(2)(2)              1                       2
26       Completion      task(2)(2)              task(63)                6                       4
28       Preemption      task(63)                task(1)(7)
29       Completion      task(1)(7)              task(63)                1                       2
30       Preemption      task(63)                task(2)(3)
32       Preemption      task(2)(3)              task(1)(8)
33       Completion      task(1)(8)              task(2)(3)              1                       2
35       Completion      task(2)(3)              task(63)                5                       4
36       Preemption      task(63)                task(1)(9)
37       Completion      task(1)(9)              task(63)                1                       2
40       Preemption      task(63)                task(1)(10)
```

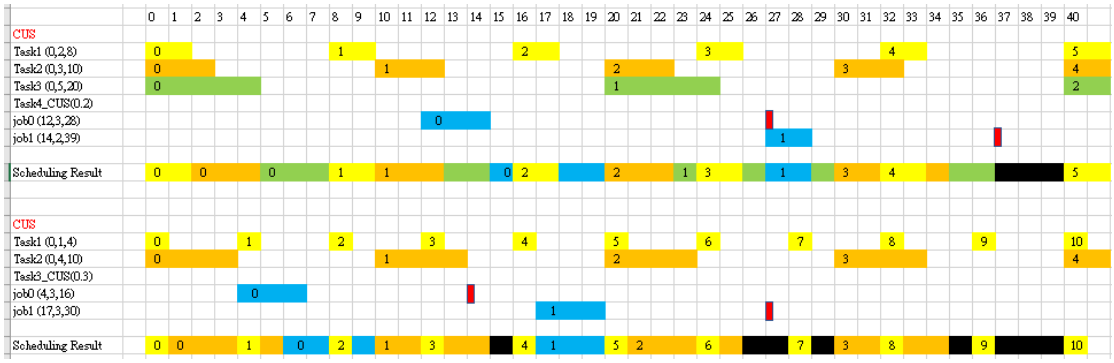
========== Task Set 2 ==========

**Periodic Task Set2 = {τ1 (0, 2, 8), τ2 (0, 3, 10), τ3 (0, 5, 20), τ4_ServerSize (0.2)}**

**Aperiodic Jobs Set2 = {j0 (12, 3, 28), j1 (14, 2, 39)}**



- 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%)




CUS 實現方式條列說明:

1. 在OS_TCB中新增變數

```
INT32U aperiodic; //非週期的工作數量
float aperiodic_serversize; //非週期的使用率
INT32U *aperiodic_arrival; //非週期的到達時間陣列指標
INT32U *aperiodic_execution; //非週期的執行時間陣列指標
INT32U *aperiodic_deadline; //非週期給定的最後期限陣列指標
INT32U aperiodic_count; //查看下個task是否要到達的變數
```

```
#define TASK4_ID 4 //定義task4的id、priority、arrival、execution、period
#define TASK4_PRIORITY 4
#define TASK4_ARRIVAL 0
#define TASK4_EXECUTION 0
#define TASK4_PERIOD 0
INT32U TASK4_aperiodic = 2; //定義非週期的工作數量
float TASK4_serversize = 0.2; //定義使用率
static INT32U TASK4_arrival[] = { 12,14 }; //依序定義工作的arrival
static INT32U TASK4_execution[] = { 3,2 }; //依序定義工作的execution
static INT32U TASK4_deadline[] = { 28,39 }; //依序定義工作的deadline
```

2. 非週期的period = execution / serversize。
3. 真正arraival的時間點 = max(上個工作deadline, 這次預設的arrival)。
4. 當完成設定的工作數量後，OS_SchedNew()就不會再把非週期的這個 task 排進去。
5. 當非週期的工作完成時，OS_TCB 內的變數會設定到下個工作變數。
6. 在 OSTimeTick()內判斷當預設的 arrival 到達時，是否在前一個 task 的 deadline 前，再顯示對應的相關字串。

# Main.c

```
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  static OS_STK Task4_STK[TASK_STACKSIZE];
69
70  #define TASK1_ID 1 //定義task1的id、priority、arrival、execution、period
71  #define TASK1_PRIORITY 1
72  #define TASK1_ARRIVAL 0
73  #define TASK1_EXECUTION 2
74  #define TASK1_PERIOD 8
75  INT32U TASK1_aperiodic = 0; //定義非週期的工作數量
76  float TASK1_serversize = 0; //定義使用率
77  static INT32U TASK1_arrival[] = { 0 }; //依序定義工作的arrival
78  static INT32U TASK1_execution[] = { 0 }; //依序定義工作的execution
79  static INT32U TASK1_deadline[] = { 0 }; //依序定義工作的deadline
80
81  #define TASK2_ID 2 //定義task2的id、priority、arrival、execution、period
82  #define TASK2_PRIORITY 2
83  #define TASK2_ARRIVAL 0
84  #define TASK2_EXECUTION 3
85  #define TASK2_PERIOD 10
86  INT32U TASK2_aperiodic = 0; //定義非週期的工作數量
87  float TASK2_serversize = 0; //定義使用率
88  static INT32U TASK2_arrival[] = { 0 }; //依序定義工作的arrival
89  static INT32U TASK2_execution[] = { 0 }; //依序定義工作的execution
90  static INT32U TASK2_deadline[] = { 0 }; //依序定義工作的deadline
91
92  #define TASK3_ID 3 //定義task3的id、priority、arrival、execution、period
93  #define TASK3_PRIORITY 3
94  #define TASK3_ARRIVAL 0
95  #define TASK3_EXECUTION 5
96  #define TASK3_PERIOD 20
97  INT32U TASK3_aperiodic = 0; //定義非週期的工作數量
98  float TASK3_serversize = 0; //定義使用率
99  static INT32U TASK3_arrival[] = { 0 }; //依序定義工作的arrival
100 static INT32U TASK3_execution[] = { 0 }; //依序定義工作的execution
101 static INT32U TASK3_deadline[] = { 0 }; //依序定義工作的deadline
102
103 #define TASK4_ID 4 //定義task4的id、priority、arrival、execution、period
104 #define TASK4_PRIORITY 4
105 #define TASK4_ARRIVAL 0
106 #define TASK4_EXECUTION 0
107 #define TASK4_PERIOD 0
108 INT32U TASK4_aperiodic = 2; //定義非週期的工作數量
109 float TASK4_serversize = 0.2; //定義使用率
110 static INT32U TASK4_arrival[] = { 12,14 }; //依序定義工作的arrival
111 static INT32U TASK4_execution[] = { 3,2 }; //依序定義工作的execution
112 static INT32U TASK4_deadline[] = { 28,39 }; //依序定義工作的deadline
```

```
169     #ifdef TASK1_ID //假如task1有被定義的話 就創建task1工作
170         Jack_OSTaskCreateExt(task1,
171             0u,
172             &Task1_STK[TASK_STACKSIZE - 1u],
173             TASK1_PRIORITY,
174             TASK1_ID,
175             &Task1_STK[0u],
176             TASK_STACKSIZE,
177             0u,
178             (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR),
179             TASK1_ARRIVAL,
180             TASK1_EXECUTION,
181             TASK1_PERIOD,
182             TASK1_aperiodic,
183             TASK1_serversize,
184             &TASK1_arrival,
185             &TASK1_execution,
186             &TASK1_deadline);
187     #endif // TASK1_ID
188
189
190     #ifdef TASK2_ID //假如task2有被定義的話 就創建task2工作
191         Jack_OSTaskCreateExt(task2,
192             0,
193             &Task2_STK[TASK_STACKSIZE - 1u],
194             TASK2_PRIORITY,
195             TASK2_ID,
196             &Task2_STK[0u],
197             TASK_STACKSIZE,
198             0u,
199             (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR),
200             TASK2_ARRIVAL,
201             TASK2_EXECUTION,
202             TASK2_PERIOD,
203             TASK2_aperiodic,
204             TASK2_serversize,
205             &TASK2_arrival,
206             &TASK2_execution,
207             &TASK2_deadline);
208     #endif // TASK2_ID
```

```
210     #ifdef TASK3_ID //假如task3有被定義的話 就創建task3工作
211         Jack_OSTaskCreateExt(task3,
212             0,
213             &Task3_STK[TASK_STACKSIZE - 1u],
214             TASK3_PRIORITY,
215             TASK3_ID,
216             &Task3_STK[0u],
217             TASK_STACKSIZE,
218             0u,
219             (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR),
220             TASK3_ARRIVAL,
221             TASK3_EXECUTION,
222             TASK3_PERIOD,
223             TASK3_aperiodic,
224             TASK3_serversize,
225             &TASK3_arrival,
226             &TASK3_execution,
227             &TASK3_deadline);
228     #endif // TASK3_ID
229
230     #ifdef TASK4_ID //假如task4有被定義的話 就創建task4工作
231         Jack_OSTaskCreateExt(task4,
232             0,
233             &Task4_STK[TASK_STACKSIZE - 1u],
234             TASK4_PRIORITY,
235             TASK4_ID,
236             &Task4_STK[0u],
237             TASK_STACKSIZE,
238             0u,
239             (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR),
240             TASK4_ARRIVAL,
241             TASK4_EXECUTION,
242             TASK4_PERIOD,
243             TASK4_aperiodic,
244             TASK4_serversize,
245             &TASK4_arrival,
246             &TASK4_execution,
247             &TASK4_deadline);
248     #endif // TASK4_ID
249
250         OSTimeSet(0); //重新歸零timetick
251
```

```c
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);
            }
        }
    }
}
```

```c
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);
            }
        }
    }
}
```

```c
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);
            }
        }
    }
}
```

```c
void task4(void* p_arg) //task4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

```
    //OS_TCB內新增變數

    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
    INT32U deadline; //EDF的最後期限
    INT32U aperiodic; //非週期的工作數量
    float aperiodic_serversize; //非週期的使用率
    INT32U *aperiodic_arrival; //非週期的到達時間陣列指標
    INT32U *aperiodic_execution; //非週期的執行時間陣列指標
    INT32U *aperiodic_deadline; //非週期給定的最後期限陣列指標
    INT32U aperiodic_count; //查看下個task是否要到達的變數
} OS_TCB;
```

# os_task.c

```
INT8U  Jack_OSTaskCreateExt(void   (*task)(void* p_arg),
   void* p_arg,
   OS_STK* ptos,
   INT8U    prio,
   INT16U   id,
   OS_STK* pbos,
   INT32U   stk_size,
   void* pext,
   INT16U   opt,
   INT32U   arrival,
   INT32U   execution,
   INT32U   period,
   INT32U   aperiodic,
   float   aperiodic_serversize,
   INT32U* aperiodic_arrival,
   INT32U* aperiodic_execution,
   INT32U* aperiodic_deadline) //在 Jack_OSTaskCreateExt 新增 OS_TCB 所需要的變數
{


   OS_STK* psp;
   INT8U       err;
#if OS_CRITICAL_METHOD == 3u 現用前置處理器區塊
#endif



#ifdef OS_SAFETY_CRITICAL_IEC61508 非現用前置處理器區塊
#endif

#if OS_ARG_CHK_EN > 0u 現用前置處理器區塊
#endif
   OS_ENTER_CRITICAL();
   if (OSIntNesting > 0u) { ... }
   if (OSTCBPrioTbl[prio] == (OS_TCB*)0) { /* Make sure task doesn't already exist at this priority  */
      OSTCBPrioTbl[prio] = OS_TCB_RESERVED;/* Reserve the priority to prevent others from doing ...  */
                                           /* ... the same thing until task is created.               */
      OS_EXIT_CRITICAL();

#if (OS_TASK_STAT_STK_CHK_EN > 0u)
      OS_TaskStkClr(pbos, stk_size, opt);                /* Clear the task stack (if needed)        */
#endif

      psp = OSTaskStkInit(task, p_arg, ptos, opt);       /* Initialize the task's stack             */
      //在 Jack_OS_TCBInit 新增 OS_TCB 所需要的變數
      err = Jack_OS_TCBInit(prio, psp, pbos, id, stk_size, pext, opt, arrival, execution, period,
          aperiodic, aperiodic_serversize, aperiodic_arrival, aperiodic_execution, aperiodic_deadline);
```

# os_core. c

```
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
```

```
2554  INT8U  Jack_OS_TCBInit(INT8U    prio,
2555      OS_STK* ptos,
2556      OS_STK* pbos,
2557      INT16U   id,
2558      INT32U   stk_size,
2559      void* pext,
2560      INT16U   opt,
2561      INT32U   arrival,
2562      INT32U   execution,
2563      INT32U   period,
2564      INT32U   aperiodic,
2565      float   aperiodic_serversize,
2566      INT32U* aperiodic_arrival,
2567      INT32U* aperiodic_execution,
2568      INT32U* aperiodic_deadline) //在 Jack_OS_TCBInit 新增 OS_TCB 所需要的變數
2569  {
```

Jack_OS_TCBInit()

```
2589
2590        ////初始化在OS_TCB內非週期的變數
2591        ptcb->aperiodic = aperiodic;
2592        ptcb->aperiodic_serversize = aperiodic_serversize;
2593        ptcb->aperiodic_arrival = aperiodic_arrival;
2594        ptcb->aperiodic_execution = aperiodic_execution;
2595        ptcb->aperiodic_deadline = aperiodic_deadline;
2596        if (aperiodic == 0) //初始化 週期工作的變數
2597        {
2598            ptcb->begin_ready_time = arrival;
2599            ptcb->response = execution;
2600            ptcb->arrival = arrival;
2601            ptcb->execution = execution;
2602            ptcb->period = period;
2603            ptcb->job_id = 0;
2604            ptcb->deadline = arrival + period;
2605        }
2606        else if (aperiodic > 0) //初始化 非週期工作的變數
2607        {
2608            ptcb->begin_ready_time = aperiodic_arrival[0];
2609            ptcb->response = aperiodic_execution[0];
2610            ptcb->arrival = aperiodic_arrival[0];
2611            ptcb->execution = aperiodic_execution[0];
2612            ptcb->job_id = 0;
2613            //計算 非週期工作 此次的task的週期時間
2614            INT32U a = aperiodic_execution[0];
2615            ptcb->period = (INT32U)((float)a / aperiodic_serversize);
2616            ptcb->deadline = aperiodic_arrival[0] + ptcb->period; //EDF的deadline 就是 到達時間加週期時間
2617            ptcb->OSTCBDly = aperiodic_arrival[0];
2618        }
2619
```

Jack_OS_TCBInit()

```
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        }
```

OSTimeTick()

```
1034        //這段是加在OSTimeTick內
1035        //從ReadyTable得到目前正在Ready的task
1036        //把現在正在Ready但卻沒有在執行的task的反應時間加一
1037        for (INT8U y = 0; y < 8; y++) //jack
1038        {
1039            for (INT8U x = 0; x < 8; x++)
1040            {
1041
1042                if ((OSRdyTbl[y] & (INT8U)1 << x) == (INT8U)1 << x) //確認是有有Ready
1043                {
1044                    INT8U prio = (y << 3u) + x;
1045                    if (OSTCBPrioTbl[prio] != OSTCBCur && prio != OS_TASK_IDLE_PRIO)
1046                    {
1047                        if (OSTimeGet() > OSTCBPrioTbl[prio]->arrival)
1048                        {
1049                            OSTCBPrioTbl[prio]->response += 1;
1050                        }
1051                    }
1052                }
1053            }
1054        }
```

OSTimeTick()

```
1056        //while查詢所有linked list的task
1057        OS_TCB* mytcb = OSTCBPrioTbl[OS_TASK_IDLE_PRIO]->OSTCBPrev;
1058        while (mytcb != (OS_TCB*)0)
1059        {
1060            //週期工作的miss deadline處理
1061            if (mytcb->aperiodic == 0)
1062            {
1063                //假設反應時間 大於週期 表示說task已經missdeadline了
1064                //這裡有兩種miss deadline處理方式
1065                if (OSTimeGet() - OSTCBCur->begin_ready_time != OSTCBCur->response)
1066                {
1067                    //第一種: while 直接卡住
1068                    /*if (OSTimeGet() > mytcb->deadline)
1069                    {
1070                        printf("%d\t MissDeadline\t Task(%d)(%d)\t\t ----------\n", OSTimeGet() - 1,
1071                            mytcb->OSTCBId, mytcb->job_id);
1072                        while (1) {}
1073                    }*/
1074
1075                    //第二種: 放棄這次工作 直接切換下個週期工作
1076                    if (OSTimeGet() == mytcb->deadline)
1077                    {
1078                        printf("%d\t MissDeadline\t Task(%d)(%d)\t\t ----------\n", OSTimeGet(),
1079                            mytcb->OSTCBId, mytcb->job_id);
1080                        mytcb->begin_ready_time = OSTimeGet();
1081                        mytcb->response = mytcb->execution;
1082                        mytcb->OSTCBCtxSwCtr = 0;
1083
1084                        //完成次數+1
1085                        mytcb->job_id += 1;
1086
1087                        mytcb->deadline = (OSTCBCur->job_id + 1) * OSTCBCur->period + OSTCBCur->arrival;
1088                    }
1089                }
1090            }
1091            //非週期工作的miss deadline處理
1092            else if(mytcb->job_id < mytcb->aperiodic)
1093            {
1094                if (OSTimeGet() > mytcb->deadline)
1095                {
1096                    printf("%d\t Aperiodic job(%d) misses deadline.\n", OSTimeGet(), mytcb->job_id);
1097                    while (1) {}
1098                }
1099            }
```

```
1101    //當非週期工作真正到達要工作時
1102    if (mytcb->aperiodic > 0 && OSTimeGet() == mytcb->arrival)
1103    {
1104        mytcb->begin_ready_time = OSTimeGet();  //begin_ready_time是task轉變為Ready的時間點
1105        mytcb->response = mytcb->aperiodic_execution[mytcb->job_id]; //反應時間等於執行時間
1106        mytcb->OSTCBCtxSwCtr = 0; //context switch次數設為0
1107    }
1108
1109    //查看非週期工作 時間已經是設定的到達時間 查看是否前面還有工作顯示相對應字串
1110    if (mytcb->aperiodic_count < mytcb->aperiodic)
1111    {
1112        if (OSTimeGet() == mytcb->aperiodic_arrival[mytcb->aperiodic_count])
1113        {
1114            //計算前次工作的deadline
1115            INT32U max = mytcb->aperiodic_arrival[mytcb->aperiodic_count];
1116            INT32U previous_deadline = mytcb->aperiodic_execution[mytcb->aperiodic_count - 1];
1117            previous_deadline = (INT32U)((float)previous_deadline / mytcb->aperiodic_serversize);
1118            previous_deadline = mytcb->aperiodic_arrival[mytcb->aperiodic_count - 1] + previous_deadline;
1119            if (max < previous_deadline)
1120                max = previous_deadline;
1121            INT32U arrival = max;
1122
1123            INT32U execution = mytcb->aperiodic_execution[mytcb->aperiodic_count];
1124            INT32U a = mytcb->aperiodic_execution[mytcb->aperiodic_count];
1125            INT32U period = (INT32U)((float)a / mytcb->aperiodic_serversize);
1126            INT32U deadline = arrival + period;
1127
1128            //前次的工作還的deadline還沒過 顯示已到達但不工作
1129            if (mytcb->aperiodic_count != 0 && mytcb->aperiodic_arrival[mytcb->aperiodic_count] < previous_deadline)
1130            {
1131                printf("%d\t Aperiodic job(%d) arrives. Do nothing.\n", OSTimeGet(), mytcb->aperiodic_count);
1132                printf("%d\t Aperiodic job(%d) sets CUS server's deadline as %d.\n", OSTimeGet(), mytcb->aperiodic_count, deadline);
1133            }
1134            else
1135            {
1136                printf("%d\t Aperiodic job(%d) arrives and sets CUS server's deadline as %d.\n", OSTimeGet(), mytcb->aperiodic_count, mytcb->deadline);
1137            }
1138            mytcb->aperiodic_count += 1;
1139        }
1140    }
1141    mytcb = mytcb->OSTCBPrev;
1142 }
```

OSTimeTick()

```
1146        ptcb = OSTCBList;                                      /* Point at first TCB in TCB list               */
1147        while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {         /* Go through all TCBs in TCB list              */
1148            OS_ENTER_CRITICAL();
1149            if (ptcb->OSTCBDly != 0u) {                        /* No, Delayed or waiting for event with TO     */
1150                ptcb->OSTCBDly--;                              /* Decrement nbr of ticks to end of delay       */
1151                if (ptcb->OSTCBDly == 0u) {                    /* Check for timeout                            */
1152
1153                    if ((ptcb->OSTCBStat & OS_STAT_PEND_ANY) != OS_STAT_RDY) {
1154                        ptcb->OSTCBStat  &= (INT8U)~(INT8U)OS_STAT_PEND_ANY;   /* Yes, Clear status flag   */
1155                        ptcb->OSTCBStatPend = OS_STAT_PEND_TO;                 /* Indicate PEND timeout    */
1156                    } else {
1157                        ptcb->OSTCBStatPend = OS_STAT_PEND_OK;
1158                    }
1159
1160                    if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == OS_STAT_RDY) {  /* Is task suspended?       */
1161                        OSRdyGrp               |= ptcb->OSTCBBitY;             /* No,  Make ready          */
1162                        OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
1163
1164                        if (ptcb->aperiodic == 0)
1165                        {
1166                            //這段是加在OSTimeTick內
1167                            //當某個task轉變為Ready的時間 設定變數數值
1168                            ptcb->begin_ready_time = OSTimeGet();  //begin_ready_time是task轉變為Ready的時間點
1169                            ptcb->response = ptcb->execution; //反應時間等於執行時間
1170                            ptcb->OSTCBCtxSwCtr = 0; //context switch次數設為0
1171                            ptcb->deadline = (ptcb->job_id + 1) * ptcb->period + ptcb->arrival; //jack
1172                        }
1173
1174                        OS_TRACE_TASK_READY(ptcb);
1175                    }
1176                }
1177            }
1178            ptcb = ptcb->OSTCBNext;                            /* Point at next TCB in TCB list                */
1179            OS_EXIT_CRITICAL();
1180        }
1181    }
1182 }
```

OSTimeTick()

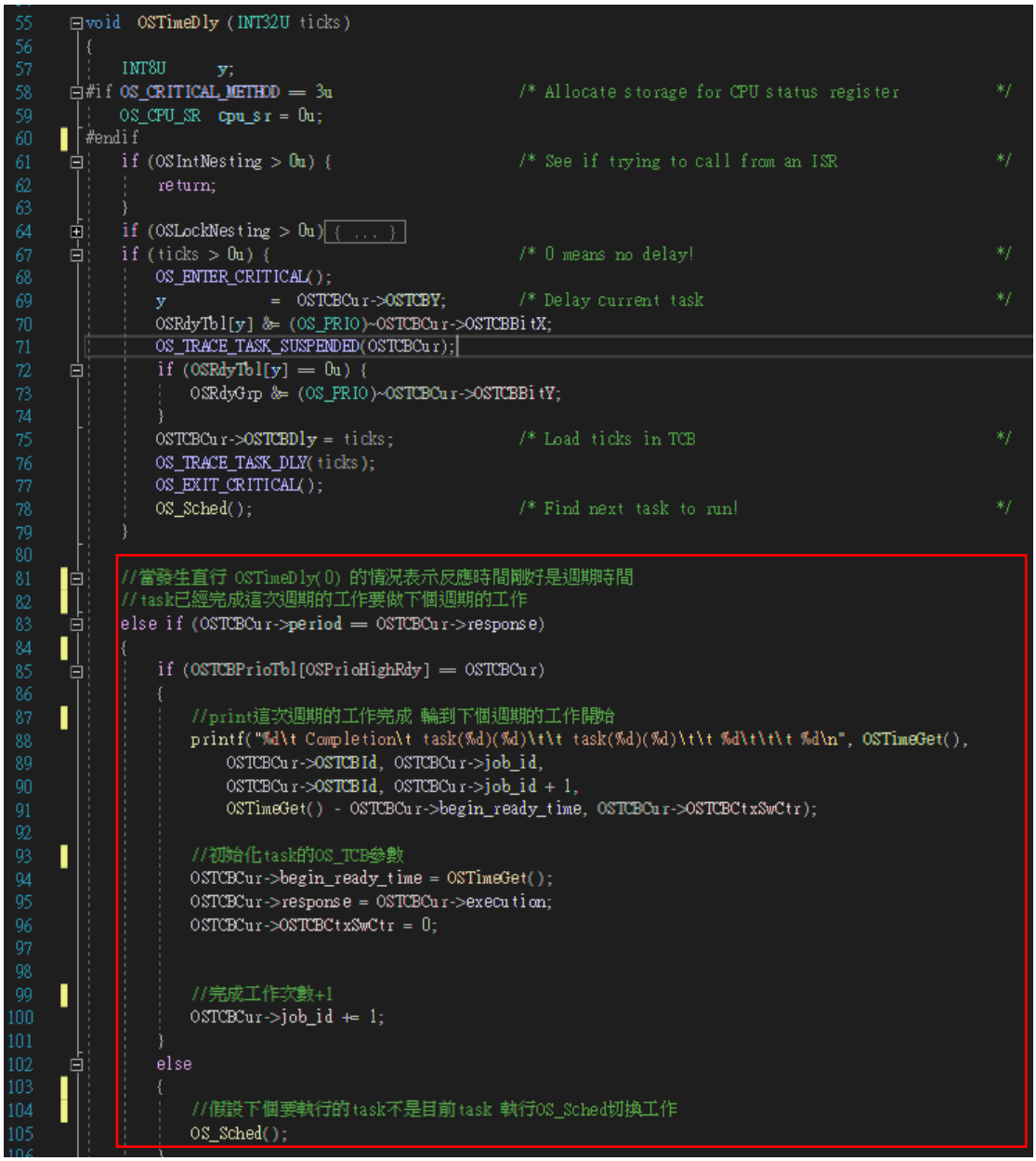
```
1988  static  void  OS_SchedNew (void)
1989  {
1990  #if OS_LOWEST_PRIO <= 63u                         /* See if we support up to 64 tasks                   */
1991      //從 OS_SchedNew 來實現EDF功能
1992      //比較所有task的deadline來決定要切換到哪個task
1993      //查看是否有除了idle task以外的task在ready
1994      OS_TCB* PrioHighTCB = OSTCBPrioTbl[OS_TASK_IDLE_PRIO];
1995      OS_TCB* mytcb = OSTCBPrioTbl[OS_TASK_IDLE_PRIO];
1996      while (mytcb != (OS_TCB*)0)
1997      {
1998
1999          if ((OSRdyTbl[mytcb->OSTCBY] & (INT8U)1 << mytcb->OSTCBX) == (INT8U)1 << mytcb->OSTCBX) //確認是有有Ready
2000          {
2001              //只計算週期工作 以及 非週期工作在已設定的工作數量內
2002              if(mytcb->aperiodic == 0 || (mytcb->job_id < mytcb->aperiodic && OSTimeGet() >= mytcb->arrival))
2003                  PrioHighTCB = mytcb;
2004          }
2005
2006          mytcb = mytcb->OSTCBPrev;
2007      }
2008      //假設沒有其他工作 將工作切換到idle task
2009      if (PrioHighTCB == OSTCBPrioTbl[OS_TASK_IDLE_PRIO])
2010      {
2011          OSPrioHighRdy = OS_TASK_IDLE_PRIO;
2012      }
2013      else
2014      {
2015          mytcb = OSTCBPrioTbl[OS_TASK_IDLE_PRIO]->OSTCBPrev;
2016          while (mytcb != (OS_TCB*)0)
2017          {
2018              if ((OSRdyTbl[mytcb->OSTCBY] & (INT8U)1 << mytcb->OSTCBX) == (INT8U)1 << mytcb->OSTCBX) //確認是有有Ready
2019              {   //只計算週期工作 以及 非週期工作在已設定的工作數量內
2020                  if (mytcb->aperiodic == 0 || mytcb->job_id < mytcb->aperiodic)
2021                  {
2022                      //比較哪個task deadline最小
2023                      if (mytcb->deadline < PrioHighTCB->deadline)
2024                      {
2025                          PrioHighTCB = mytcb;
2026                      }
2027                      //deadline相同的話 取id比較小的task
2028                      else if (mytcb->deadline == PrioHighTCB->deadline && mytcb->OSTCBId < PrioHighTCB->OSTCBId)
2029                      {
2030                          PrioHighTCB = mytcb;
2031                      }
2032                  }
2033              }
2034              mytcb = mytcb->OSTCBPrev;
2035          }
2036          OSPrioHighRdy = PrioHighTCB->OSTCBPrio;
```

```
1876  void  OS_Sched (void)
1877  {
1878  #if OS_CRITICAL_METHOD == 3u                       /* Allocate storage for CPU status register     */
1879      OS_CPU_SR  cpu_sr = 0u;
1880  #endif
1881
1882
1883      OS_ENTER_CRITICAL();
1884      if (OSIntNesting == 0u) {                      /* Schedule only if all ISRs done and ...       */
1885          if (OSLockNesting == 0u) {                 /* ... scheduler is not locked                  */
1886              OS_SchedNew();
1887              OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
1888              if (OSPrioHighRdy != OSPrioCur) {          /* No Ctx Sw if current task is highest rdy     */
1889
1890                  if (OSTimeGet() > OSTCBCur->arrival)
1891                  {
1892                      //週期工作的處理方式
1893                      if (OSTCBCur->aperiodic == 0)
1894                      {
1895                          OSTCBCur->OSTCBCtxSwCtr++; //OSTCBCtxSwCtr加一
1896
1897                          //task完成之後會執行OS_Sched所以在這裡print完成
1898                          //假設是task完成後切入idle task，print的方式會不一樣
1899                          if (OSTCBHighRdy->OSTCBPrio == OS_TASK_IDLE_PRIO)
1900                              printf("%d\t Completion\t task(%d)(%d)\t\t task(%d)  \t\t %d\t\t\t\t %d\n", OSTimeGet(),
1901                                  OSTCBCur->OSTCBId, OSTCBCur->job_id,
1902                                  OS_TASK_IDLE_PRIO,
1903                                  OSTimeGet() - OSTCBCur->begin_ready_time, OSTCBCur->OSTCBCtxSwCtr);
1904                          else
1905                              printf("%d\t Completion\t task(%d)(%d)\t\t task(%d)(%d)\t\t %d\t\t\t\t %d\n", OSTimeGet(),
1906                                  OSTCBCur->OSTCBId, OSTCBCur->job_id,
1907                                  OSTCBHighRdy->OSTCBId, OSTCBHighRdy->job_id,
1908                                  OSTimeGet() - OSTCBCur->begin_ready_time, OSTCBCur->OSTCBCtxSwCtr);
1909
1910                          //task完成所以將變數初始化
1911                          OSTCBCur->begin_ready_time = OSTimeGet();
1912                          OSTCBCur->response = OSTCBCur->execution;
1913                          OSTCBCur->OSTCBCtxSwCtr = 0;
1914
1915                          //完成次數+1
1916                          OSTCBCur->job_id += 1;
1917
1918                          OSTCBCur->deadline = (OSTCBCur->job_id + 1) * OSTCBCur->period + OSTCBCur->arrival;
1919                      }
```

```
1920      //非週期工作的處理方式
1921      else if (OSTCBCur->aperiodic > 0 && OSTCBCur->job_id < OSTCBCur->aperiodic)
1922      {
1923          OSTCBCur->OSTCBCtxSwCtr++; //OSTCBCtxSwCtr加一
1924
1925          printf("%d\t Aperiodic job(%d) is finished.\n", OSTimeGet(), OSTCBCur->job_id);
1926          printf("%d\t Completion\t task(%d)(%d)\t\t task(%d)(%d)\t\t %d\t\t\t\t %d\n", OSTimeGet(),
1927              OSTCBCur->OSTCBId, OSTCBCur->job_id,
1928              OSTCBHighRdy->OSTCBId, OSTCBHighRdy->job_id,
1929              OSTimeGet() - OSTCBCur->begin_ready_time, OSTCBCur->OSTCBCtxSwCtr);
1930
1931          OSTCBCur->job_id += 1;
1932          if (OSTCBCur->job_id < OSTCBCur->aperiodic) //在設定的工作數量內
1933          {
1934              OSTCBCur->begin_ready_time = OSTCBCur->aperiodic_arrival[OSTCBCur->job_id]; //初始化在OS_TCB內新增的變數
1935              OSTCBCur->response = OSTCBCur->aperiodic_execution[OSTCBCur->job_id];
1936              OSTCBCur->OSTCBCtxSwCtr = 0;
1937
1938              //計算上個工作的deadline時間 並比較 此次預設的arrival跟上次的deadline哪個比較大 設定task真正要arrival的時間
1939              INT32U max = OSTCBCur->aperiodic_arrival[OSTCBCur->job_id];
1940              INT32U previous_deadline = OSTCBCur->aperiodic_execution[OSTCBCur->job_id-1];
1941              previous_deadline = (INT32U)((float)previous_deadline / OSTCBCur->aperiodic_serversize);
1942              previous_deadline = OSTCBCur->arrival + previous_deadline;
1943              if (max < previous_deadline)
1944                  max = previous_deadline;
1945              OSTCBCur->arrival = max;
1946
1947              OSTCBCur->execution = OSTCBCur->aperiodic_execution[OSTCBCur->job_id];
1948              INT32U a = OSTCBCur->aperiodic_execution[OSTCBCur->job_id];
1949              OSTCBCur->period = (INT32U)((float)a / OSTCBCur->aperiodic_serversize);
1950              OSTCBCur->deadline = OSTCBCur->arrival + OSTCBCur->period;
1951
1952          }
1953      }
```

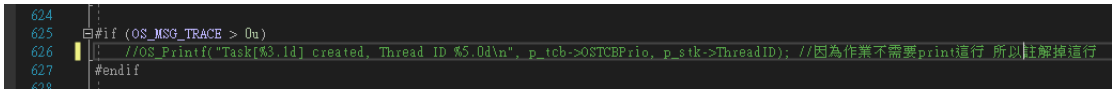
```
695  void  OSIntExit (void)
696  {
697  #if OS_CRITICAL_METHOD == 3u                          /* Allocate storage for CPU status register */
698      OS_CPU_SR  cpu_sr = 0u;
699  #endif
700
701      if (OSRunning == OS_TRUE) {
702          OS_ENTER_CRITICAL();
703          if (OSIntNesting > 0u) {                      /* 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                  //在task快要完成時 先切換到下個deadline時間 這樣就能去判斷下個要換哪個task執行
709                  if (OSTimeGet() - OSTCBCur->begin_ready_time == OSTCBCur->response)
710                  {
711                      OSTCBCur->deadline = (OSTCBCur->job_id + 2) * OSTCBCur->period + OSTCBCur->arrival;
712                  }
713                  OS_SchedNew();
714                  //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
715                  //迴避掉這次切換讓原本的task先做完
716                  if (OSTimeGet() - OSTCBCur->begin_ready_time != OSTCBCur->response)
717                  {
718
719                      OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
720                      //printf("OSPrioHighRdy: %d , OSPrioCur: %d\n", OSPrioHighRdy, OSPrioCur);
721                      if (OSPrioHighRdy != OSPrioCur) {          /* No Ctx Sw if current task is highest rdy */
722
723                          OSTCBCur->OSTCBCtxSwCtr++; //OSTCBCtxSwCtr加一
724
725                          //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
726                          //假設是idle task被搶占，print的方式會不一樣
727                          if (OSTCBCur->OSTCBPrio == OS_TASK_IDLE_PRIO)
728                              printf("%d\t Preemption\t task(%d)  \t\t task(%d)(%d)\n", OSTimeGet(),
729                                  OS_TASK_IDLE_PRIO,
730                                  OSTCBHighRdy->OSTCBId, OSTCBHighRdy->job_id);
731                          else
732                              printf("%d\t Preemption\t task(%d)(%d)\t\t task(%d)(%d)\n", OSTimeGet(),
733                                  OSTCBCur->OSTCBId, OSTCBCur->job_id,
734                                  OSTCBHighRdy->OSTCBId, OSTCBHighRdy->job_id);
735
```

# os_time.c



```
void  OSTimeDly (INT32U ticks)
{
    INT8U      y;
#if OS_CRITICAL_METHOD == 3u                    /* Allocate storage for CPU status register           */
    OS_CPU_SR  cpu_sr = 0u;
#endif
    if (OSIntNesting > 0u) {                    /* See if trying to call from an ISR                  */
        return;
    }
    if (OSLockNesting > 0u) { ... }
    if (ticks > 0u) {                           /* 0 means no delay!                                  */
        OS_ENTER_CRITICAL();
        y             =  OSTCBCur->OSTCBY;      /* Delay current task                                 */
        OSRdyTbl[y] &= (OS_PRIO)~OSTCBCur->OSTCBBitX;
        OS_TRACE_TASK_SUSPENDED(OSTCBCur);
        if (OSRdyTbl[y] == 0u) {
            OSRdyGrp &= (OS_PRIO)~OSTCBCur->OSTCBBitY;
        }
        OSTCBCur->OSTCBDly = ticks;             /* Load ticks in TCB                                  */
        OS_TRACE_TASK_DLY(ticks);
        OS_EXIT_CRITICAL();
        OS_Sched();                             /* Find next task to run!                             */
    }

    //當發生直行 OSTimeDly(0) 的情況表示反應時間剛好是週期時間
    //task已經完成這次週期的工作要做下個週期的工作
    else if (OSTCBCur->period == OSTCBCur->response)
    {
        if (OSTCBPrioTbl[OSPrioHighRdy] == OSTCBCur)
        {
            //print這次週期的工作完成 輪到下個週期的工作開始
            printf("%d\t Completion\t task(%d)(%d)\t\t task(%d)(%d)\t\t %d\t\t\t\t %d\n", OSTimeGet(),
                OSTCBCur->OSTCBId, OSTCBCur->job_id,
                OSTCBCur->OSTCBId, OSTCBCur->job_id + 1,
                OSTimeGet() - OSTCBCur->begin_ready_time, OSTCBCur->OSTCBCtxSwCtr);

            //初始化task的OS_TCB參數
            OSTCBCur->begin_ready_time = OSTimeGet();
            OSTCBCur->response = OSTCBCur->execution;
            OSTCBCur->OSTCBCtxSwCtr = 0;


            //完成工作次數+1
            OSTCBCur->job_id += 1;
        }
        else
        {
            //假設下個要執行的task不是目前task 執行OS_Sched切換工作
            OS_Sched();
        }
```

# os_cpu_c.c



```
#if (OS_MSG_TRACE > 0u)
    //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
#endif
```

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