RTOS-PA1

學號：M11007326 姓名：黃鈞臨

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

- The screenshot results.(10%)
  - Task Set 1 ={τ1 (1, 1, 2, 4), τ2 (2, 0, 4, 10)}

```
OSTick    created, Thread ID  3716
Task[ 63] created, TCB Address  24e640
------After TCB[63] being linked------
Previous TCB point to address        0
Current  TCB point to address   24e640
Next     TCB point to address        0

The file 'TaskSet.txt' was opened
Task[  1] created, TCB Address  24e6a8
------After TCB[ 1] being linked------
Previous TCB point to address        0
Current  TCB point to address   24e6a8
Next     TCB point to address   24e640

Task[  2] created, TCB Address  24e710
------After TCB[ 2] being linked------
Previous TCB point to address        0
Current  TCB point to address   24e710
Next     TCB point to address   24e6a8

================TCB linked list================
Task   Prev_TCB_addr   TCB_addr   Next_TCB_addr

 2              0       24e710          24e6a8
 1          24e710       24e6a8          24e640
63          24e6a8       24e640               0
```

  - Task Set 2 ={τ1 (1, 3, 4, 14), τ2 (2, 0, 2, 8), τ3 (3, 0, 4, 10), τ4 (4, 24, 2, 12)}

```
OSTick    created, Thread ID 20920
Task[ 63] created, TCB Address  24e640
------After TCB[63] being linked------
Previous TCB point to address        0
Current  TCB point to address   24e640
Next     TCB point to address        0

The file 'TaskSet.txt' was opened
Task[  2] created, TCB Address  24e6a8
------After TCB[ 2] being linked------
Previous TCB point to address        0
Current  TCB point to address   24e6a8
Next     TCB point to address   24e640

Task[  3] created, TCB Address  24e710
------After TCB[ 3] being linked------
Previous TCB point to address        0
Current  TCB point to address   24e710
Next     TCB point to address   24e6a8

Task[  4] created, TCB Address  24e778
------After TCB[ 4] being linked------
Previous TCB point to address        0
Current  TCB point to address   24e778
Next     TCB point to address   24e710
```

```
Task[  1] created, TCB Address  24e7e0
------After TCB[ 1] being linked------
Previous TCB point to address        0
Current  TCB point to address   24e7e0
Next     TCB point to address   24e778

================TCB linked list================
Task   Prev_TCB_addr   TCB_addr   Next_TCB_addr

 1              0       24e7e0          24e778
 4          24e7e0       24e778          24e710
 3          24e778       24e710          24e6a8
 2          24e710       24e6a8          24e640
63          24e6a8       24e640               0
```

- Task Set 3 ={τ1 (1, 2, 2, 10), τ2 (2, 1, 1, 5), τ3 (3, 0, 8, 15)}

```
OSTick    created, Thread ID 21332
Task[ 63] created, TCB Address  1ee640
------After TCB[63] being linked------
Previous TCB point to address        0
Current  TCB point to address   1ee640
Next     TCB point to address        0

The file 'TaskSet.txt' was opened
```

```
Task[  2] created, TCB Address  1ee6a8
------After TCB[ 2] being linked------
Previous TCB point to address        0
Current  TCB point to address   1ee6a8
Next     TCB point to address   1ee640

Task[  1] created, TCB Address  1ee710
------After TCB[ 1] being linked------
Previous TCB point to address        0
Current  TCB point to address   1ee710
Next     TCB point to address   1ee6a8

Task[  3] created, TCB Address  1ee778
------After TCB[ 3] being linked------
Previous TCB point to address        0
Current  TCB point to address   1ee778
Next     TCB point to address   1ee710

================TCB linked list================
Task   Prev_TCB_addr   TCB_addr   Next_TCB_addr

 3              0       1ee778          1ee710
 1          1ee778       1ee710          1ee6a8
 2          1ee710       1ee6a8          1ee640
63          1ee6a8       1ee640               0
```

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

```c
OSTaskCtr++;                                    /* Increment the #tasks counter          */
OS_TRACE_TASK_READY(ptcb);
if (ptcb->OSTCBPrio == 63) {
    printf("Task[%3.1d] created, TCB Address  %x\n", ptcb->OSTCBPrio, ptcb);
    printf("------After TCB[%2.1d] being linked------\n", ptcb->OSTCBPrio);
    printf("Previous TCB point to address %8x\n", ptcb->OSTCBPrev);
    printf("Current  TCB point to address %8x\n", ptcb);
    printf("Next     TCB point to address %8x\n\n", ptcb->OSTCBNext);
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
    {
        fprintf(Output_fp, "Task[%3.1d] created, TCB Address  %x\n", ptcb->OSTCBPrio, ptcb);
        fprintf(Output_fp, "------After TCB[%2.1d] being linked------\n", ptcb->OSTCBPrio);
        fprintf(Output_fp, "Previous TCB point to address %8x\n", ptcb->OSTCBPrev);
        fprintf(Output_fp, "Current  TCB point to address %8x\n", ptcb);
        fprintf(Output_fp, "Next     TCB point to address %8x\n\n", ptcb->OSTCBNext);
        fclose(Output_fp);
    }
}
else {
    printf("Task[%3.1d] created, TCB Address  %x\n", ptcb->OSTCBId, ptcb);
    printf("------After TCB[%2.1d] being linked------\n", ptcb->OSTCBId);
    printf("Previous TCB point to address %8x\n", ptcb->OSTCBPrev);
    printf("Current  TCB point to address %8x\n", ptcb);
    printf("Next     TCB point to address %8x\n\n", ptcb->OSTCBNext);
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
    {
        fprintf(Output_fp, "Task[%3.1d] created, TCB Address  %x\n", ptcb->OSTCBId, ptcb);
        fprintf(Output_fp, "------After TCB[%2.1d] being linked------\n", ptcb->OSTCBId);
        fprintf(Output_fp, "Previous TCB point to address %8x\n", ptcb->OSTCBPrev);
        fprintf(Output_fp, "Current  TCB point to address %8x\n", ptcb);
        fprintf(Output_fp, "Next     TCB point to address %8x\n\n", ptcb->OSTCBNext);
        fclose(Output_fp);
    }
}
OS_EXIT_CRITICAL();
return (OS_ERR_NONE);
}
OS_EXIT_CRITICAL();
return (OS_ERR_TASK_NO_MORE_TCB);
}
```

在 OS_TCBlint() 接近尾端的地方去 printf 每一個任務的位置資訊，主要是任務被創建時會走這邊，而且一個任務也只會進來一次，if 跟 else只是用來區別idle任務與一般的任務。

os_task.c | os_core.c | main.c | TaskSet.txt | app_hooks.c | ucos_ii.h | os_cpu_c.c

OS2 | (全域範圍) | OSStart(void)

```c
    printf("==============TCB linked list==============\n");
    printf("Task   Prev_TCB_addr   TCB_addr   Next_TCB_addr\n\n");
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
    {
        fprintf(Output_fp, "==============TCB linked list==============\n");
        fprintf(Output_fp, "Task   Prev_TCB_addr   TCB_addr   Next_TCB_addr\n\n");
        fclose(Output_fp);
    }
    while(ptcb != 0){
        if (ptcb->OSTCBPrio == 63) {
            printf("%2d     %11x       %6x     %11x  \n", ptcb->OSTCBPrio, ptcb->OSTCBPrev, ptcb, ptcb->OSTCBNext);
            if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
            {
                fprintf(Output_fp, "%2d     %11x       %6x     %11x  \n", ptcb->OSTCBPrio, ptcb->OSTCBPrev, ptcb, ptcb->OSTCBNext);
                fclose(Output_fp);
            }
            ptcb = ptcb->OSTCBNext;
        }
        else {
            printf("%2d     %11x       %6x     %11x  \n", ptcb->OSTCBId, ptcb->OSTCBPrev, ptcb, ptcb->OSTCBNext);
            ptcb = ptcb->OSTCBNext;
        }
    }

    OS_SchedNew();                          /* Find highest priority's task priority number   */
    OSPrioCur     = OSPrioHighRdy;
    OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy]; /* Point to highest priority task ready to run    */
    OSTCBCur      = OSTCBHighRdy;

    printf("\nTick   Event         CurrentTask ID   NextTask ID   ");
    printf("ResponseTime  #of ContextSwitch  PreemptionTime  OSTimeDly\n");
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
    {
        fprintf(Output_fp, "\nTick   Event         CurrentTask ID   NextTask ID   ");
        fprintf(Output_fp, "ResponseTime  #of ContextSwitch  PreemptionTime  OSTimeDly\n");
        fclose(Output_fp);
    }
    OSStartHighRdy();                       /* Execute target specific code to start task     */
}
}
```

接著在 OSStart() 裡面去 printf TCBlist 的訊息，因為在main裡面可以觀察到 OSStart() 會在任務建完後才執行，所以在這邊能夠正確的 printf 出全部已被建好的任務資訊，因為TCBlist最後會指向0，因此就使用 while 執行到指向0為止。

[PART II] RM Scheduler Implementation [80%]

- The correctness of schedule results of examples. Note the testing task set might not be the same as the given example task set. (40%)

  - Task Set 1 ={τ1 (1, 1, 2, 4), τ2 (2, 0, 4, 10)}

Tick	Event	CurrentTask ID	NextTask ID	ResponseTime	#of ContextSwitch	PreemptionTime	OSTimeDly
1	Preemption	task(2)(0)	task(1)(0)				
3	Completion	task(1)(0)	task(2)(0)	2	2	0	2
5	Preemption	task(2)(0)	task(1)(1)				
7	Completion	task(1)(1)	task(2)(0)	2	2	0	2
8	Completion	task(2)(0)	task(63)	8	5	4	2
9	Preemption	task(63)	task(1)(2)				
11	Completion	task(1)(2)	task(2)(1)	2	2	0	2
13	Preemption	task(2)(1)	task(1)(3)				
15	Completion	task(1)(3)	task(2)(1)	2	2	0	2
17	Completion	task(2)(1)	task(1)(4)	7	4	3	3
19	Completion	task(1)(4)	task(63)	2	2	0	2
20	Preemption	task(63)	task(2)(2)				
21	Preemption	task(2)(2)	task(1)(5)				
23	Completion	task(1)(5)	task(2)(2)	2	2	0	2
25	Preemption	task(2)(2)	task(1)(6)				
27	Completion	task(1)(6)	task(2)(2)	2	2	0	2
28	Completion	task(2)(2)	task(63)	8	6	4	2
29	Preemption	task(63)	task(1)(7)				

  - Task Set 2 ={τ1 (1, 3, 4, 14), τ2 (2, 0, 2, 8), τ3 (3, 0, 4, 10), τ4 (4, 24, 2, 12)}

Tick	Event	CurrentTask ID	NextTask ID	ResponseTime	#of ContextSwitch	PreemptionTime	OSTimeDly
2	Completion	task(2)(0)	task(3)(0)	2	1	0	6
6	Completion	task(3)(0)	task(1)(0)	6	2	2	4
8	Preemption	task(1)(0)	task(2)(1)				
10	Completion	task(2)(1)	task(3)(1)	2	2	0	6
14	Completion	task(3)(1)	task(1)(0)	4	2	0	6
16	Completion	task(1)(0)	task(2)(2)	13	4	9	1
18	Completion	task(2)(2)	task(1)(1)	2	2	0	6
20	Preemption	task(1)(1)	task(3)(2)				
24	Completion	task(3)(2)	task(2)(3)	4	2	0	6
26	Completion	task(2)(3)	task(4)(0)	2	2	0	6
28	Completion	task(4)(0)	task(1)(1)	4	2	2	8
30	Completion	task(1)(1)	task(3)(3)	13	4	9	1

  - Task Set 3 ={τ1 (1, 2, 2, 10), τ2 (2, 1, 1, 5), τ3 (3, 0, 8, 15)}

Tick	Event	CurrentTask ID	NextTask ID	ResponseTime	#of ContextSwitch	PreemptionTime	OSTimeDly
1	Preemption	task(3)(0)	task(2)(0)				
2	Completion	task(2)(0)	task(1)(0)	1	2	0	4
4	Completion	task(1)(0)	task(3)(0)	2	2	0	8
6	Preemption	task(3)(0)	task(2)(1)				
7	Completion	task(2)(1)	task(3)(0)	1	2	0	4
11	Preemption	task(3)(0)	task(2)(2)				
12	Completion	task(2)(2)	task(1)(1)	1	2	0	4
14	Completion	task(1)(1)	task(3)(0)	2	2	0	8
15	Completion	task(3)(0)	task(3)(1)	15	6	7	
16	Preemption	task(3)(1)	task(2)(3)				
17	Completion	task(2)(3)	task(3)(1)	1	2	0	4
21	Preemption	task(3)(1)	task(2)(4)				
22	Completion	task(2)(4)	task(1)(2)	1	2	0	4
24	Completion	task(1)(2)	task(3)(1)	2	2	0	8
26	Preemption	task(3)(1)	task(2)(5)				
27	Completion	task(2)(5)	task(3)(1)	1	2	0	4
28	Completion	task(3)(1)	task(63)	13	7	5	2
30	Preemption	task(63)	task(3)(2)				

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

```
os_task.c | os_core.c | main.c | TaskSet.txt | app_hooks.c* | ucos_ii.h | os_cpu_c.c
OS2 | (全域範圍) | InputFile()

                i++;
            }
            j++;
        }

        fclose(fp);
        for (int p = 1; p < TASK_NUMBER; p++) {
            task_para_set key = TaskParameter[p];
            int sp = p - 1;
            while (key.TaskPeriodic < TaskParameter[sp].TaskPeriodic && sp >= 0) {
                TaskParameter[sp + 1] = TaskParameter[sp];
                sp--;
            }
            TaskParameter[sp + 1] = key;
        }
        for (int p = 0; p < TASK_NUMBER; p++) {
            TaskParameter[p].TaskPriority = p;
        }
        /*read file*/

    }
```

在這次的作業要使用RM scheduler 來做排程，在這個排程中週期越短的任務優先權會越高，因此在 app_hooks.c 裡讀取檔案的地方，在讀取完txt檔內的資訊後，先用一個迴圈做任務週期的 insertion sort，將週期短的任務擺在靠前的位置，之後再用一個迴圈根據位置給予任務優先權。

```
OS2 | (全域範圍) | OS_TCBInit(INT8U prio, OS_STK * ptos, OS_
#if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
    INT8U       j;
#endif
#endif


    OS_ENTER_CRITICAL();
    ptcb = OSTCBFreeList;                                   /* Get a free TCB from the free TCB list
    if (ptcb != (OS_TCB *)0) {
        OSTCBFreeList           = ptcb->OSTCBNext;          /* Update pointer to free TCB list
        OS_EXIT_CRITICAL();
        ptcb->OSTCBStkPtr       = ptos;                     /* Load Stack pointer in TCB
        ptcb->OSTCBPrio         = prio;                     /* Load task priority into TCB
        ptcb->OSTCBStat         = OS_STAT_RDY;              /* Task is ready to run
        ptcb->OSTCBStatPend     = OS_STAT_PEND_OK;          /* Clear pend status
      //ptcb->OSTCBDly          = 0u;                       /* Task is not delayed
        ptcb->OSTCBDly          = ArriveTime;
#if OS_TASK_CREATE_EXT_EN > 0u
        ptcb->OSTCBExtPtr       = pext;                     /* Store pointer to TCB extension
        ptcb->OSTCBStkSize      = stk_size;                 /* Store stack size
        ptcb->OSTCBStkBottom    = pbos;                     /* Store pointer to bottom of stack
        ptcb->OSTCBOpt          = opt;                      /* Store task options
        ptcb->OSTCBId           = id;                       /* Store task ID
        ptcb->OSTCBExecutionTime = ExecutionTime;
        ptcb->OSTCBArriveTime   = ArriveTime;

        ptcb->OSTCBStartime     = 0uL;
        ptcb->OSTCBContexSwitch = 0uL;
        ptcb->OSTCBJobNum       = 0uL;
        ptcb->OSTCBWorkTime     = 0uL;
```

在OS_TCBlint()中加入一些需要用到的參數，OSTCBExecutionTime 代表任務的執行時間、OSTCBArriveTime 代表任務的到達時間、OSTCBStartime代表任務真正開始執行的時間、OSTCBContexSwitch代表任務的context switch次數、OSTCBJobNum代表任務已執行次數、OSTCBWorkTime代表任務使用CPU的時間。

```c
for (n = 0; n < TASK_NUMBER; n++) {
    OSTaskCreateExt(task1,
        &TaskParameter[n],
        &Task_STK[n][TASK_STACKSIZE - 1],
        TaskParameter[n].TaskPriority,
        TaskParameter[n].TaskID,
        &Task_STK[n][0],
        TASK_STACKSIZE,
        &TaskParameter[n],
        (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR),
        TaskParameter[n].TaskExecutionTime,
        TaskParameter[n].TaskArriveTime);
}
```

在main裡面的 OStaskCreateExt() 讀取到達時間以及執行時間兩個參數。

os_task.c | os_core.c | main.c | TaskSet.txt | app_hooks.c | ucos_ii.h | os_cpu_c.c

OS2 | (全域範圍) | OSTimeTick(void)

```c
#if OS_TIME_TICK_HOOK_EN > 0u
    OSTimeTickHook();                                   /* Call user definable hook
#endif
#if OS_TIME_GET_SET_EN > 0u
    OS_ENTER_CRITICAL();                                /* Update the 32-bit tick counter
    OSTime++;
    OSTCBCur->OSTCBWorkTime++;
    OS_TRACE_TICK_INCREMENT(OSTime);
    OS_EXIT_CRITICAL();
```

在 OSTimeTick() 裡面，當時間增加時，紀錄Task使用cpu時間的變數 WorkTime 也會跟著加一。

```c
printf("task(%2d)(%2d)   ", OSTCBCur->OSTCBId, OSTCBCur->OSTCBJobNum);
if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
{
    fprintf(Output_fp, "%2d     Completion      ", OSTimeGet());
    fprintf(Output_fp, "task(%2d)(%2d)   ", OSTCBCur->OSTCBId, OSTCBCur->OSTCBJobNum);
    fclose(Output_fp);
}
if (OSTCBHighRdy->OSTCBPrio == 63) {
    printf("task(%2d)%14d", OSTCBHighRdy->OSTCBPrio, OSTimeGet()-OSTCBCur->OSTCBPeriodic * (OSTCBCur->OSTCBJobNum) - OSTCBCur->OSTCBArriveTime);
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
    {
        fprintf(Output_fp, "task(%2d)%14d", OSTCBHighRdy->OSTCBPrio, OSTimeGet() - OSTCBCur->OSTCBPeriodic * (OSTCBCur->OSTCBJobNum) - OSTCBCur->OSTCBArriveTime);
        fclose(Output_fp);
    }
}
else {
    printf("task(%2d)(%2d)%10d", OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBJobNum, OSTimeGet() - OSTCBCur->OSTCBPeriodic * (OSTCBCur->OSTCBJobNum) - OSTCBCur->OSTCBArriveTime);
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0)
    {
        fprintf(Output_fp, "task(%2d)(%2d)%10d", OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBJobNum, OSTimeGet() - OSTCBCur->OSTCBPeriodic * (OSTCBCur->OSTCBJobNum) - OSTCBCur->OSTCBArriveTi
        fclose(Output_fp);
    }
}
```

若任務在OS_Sched()進行contextSwitch，則 printf 任務完成的訊息，在這個部分需要注意的是 ResponseTime，計算ResponseTime的算法是：
系統時間－當前的任務週期 × 任務已執行次數－任務的到達時間

```
if (OSTCBHighRdy->OSTCBId != OSTCBCur->OSTCBId) {
    OSTCBCur->OSTCBContexSwitch++;
    OSTCBHighRdy->OSTCBContexSwitch++;
}
```

接著把當前和下一個任務的contextSwitch次數都加一，而若當前與下一個任務是相同的話則不用加。

```
printf("%15d", OSTCBCur->OSTCBContexSwitch);
printf("%15d", OSTimeGet() - OSTCBCur->OSTCBPeriodic * (OSTCBCur->OSTCBJobNum) - OSTCBCur->OSTCBArriveTime - OSTCBCur->OSTCBExecutionTime);
printf("%15d\n", OSTCBCur->OSTCBPeriodic * (OSTCBCur->OSTCBJobNum + 1) + OSTCBCur->OSTCBArriveTime - OSTimeGet());
```

接著在下方 printf ContextSwitch次數、PreemptionTime 和 OSTimeDly

PreemptionTime 的計算方式：
系統時間－當前的任務週期 × 任務已執行次數－任務的到達時間－任務的執行時間
OSTimeDlay 的計算方式：
當前的任務週期 × (任務已執行次數+1)＋任務的到達時間－系統時間

```
OSTCBCur->OSTCBJobNum++;
OSTCBCur->OSTCBWorkTime = 0;
```

在 printf 完資訊後，把已執行次數+1、任務使用cpu的時間歸零。

```
static  void  OS_SchedNew (void)
{
#if OS_LOWEST_PRIO <= 63u                        /* See if we support up to 64 tasks                    */
    INT8U   y;


    y             = OSUnMapTbl[OSRdyGrp];
    OSPrioHighRdy = (INT8U)((y << 3u) + OSUnMapTbl[OSRdyTbl[y]]);
    if (OSTCBCur != NULL && OSTCBCur->OSTCBPrio !=63) {
        if (OSTimeGet() >= OSTCBCur->OSTCBPeriodic * (OSTCBCur->OSTCBJobNum + 1) + OSTCBCur->OSTCBArriveTime)
            if (OSTCBCur->OSTCBWorkTime == OSTCBCur->OSTCBExecutionTime) {
```

在 OS_SchedNew() 裡面，處理遇到 Deadline 的任務，先判斷任務的使用CPU時間是否已經等於執行時間，如果等於的話就把任務 Sched 給當前高優先權任務，作法與 OS_sched() 相同。

```
else {
    printf("%2d     MissDeadline     ", OSTimeGet());
    printf("task(%2d)(%2d)   \n", OSTCBCur->OSTCBId, OSTCBCur->OSTCBJobNum);
    if (OSTCBHighRdy->OSTCBId != OSTCBCur->OSTCBId) {
        OSTCBCur->OSTCBContexSwitch++;
        OSTCBHighRdy->OSTCBContexSwitch++;
    }
    OSTCBCur->OSTCBContexSwitch = 0;
    OSTCBCur->OSTCBWorkTime = 0;
    OSTCBCur->OSTCBJobNum++;
}
```

但若不等於的話就判斷任務MissDeadline，直接把當前任務跳過。

```c
void  OSIntExit (void)
{
#if OS_CRITICAL_METHOD == 3u                          /* Allocate storage for CPU status register
    OS_CPU_SR  cpu_sr = 0u;
#endif



    if (OSRunning == OS_TRUE) {
        OS_ENTER_CRITICAL();
        if (OSIntNesting > 0u) {                      /* Prevent OSIntNesting from wrapping
            OSIntNesting--;
        }

        if (OSIntNesting == 0u) {                     /* Reschedule only if all ISRs complete ...
            if (OSLockNesting == 0u) {                /* ... and not locked.
                if (OSTCBCur->OSTCBWorkTime != OSTCBCur->OSTCBExecutionTime) {
                    OS_SchedNew();
                }
```

在OSIntExit()裡面，在 SchedNew 之前先判斷當前任務是否會在這個 tick 結束，直接排有可能造成當前任務已經結束但被搶佔的情況。

```c
printf("%2d      Preemption       ", OSTimeGet());
if (OSTCBCur->OSTCBPrio == 63) {
    printf("task(%2d)        ", OSPrioCur);
}
else {
    printf("task(%2d)(%2d)    ", OSTCBCur->OSTCBId, OSTCBCur->OSTCBJobNum);
}
printf("task(%2d)(%2d)\n", OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBJobNum)

OSTCBCur->OSTCBContexSwitch++;
OSTCBHighRdy->OSTCBContexSwitch++;

OSIntCtxSw();                          /* Perform interrupt level ctx switch
```

若任務透過中斷contextSwitch 的話，printf Preemption訊息，並把切換次數加一。

```
void task1(void* p_arg) {
    task_para_set* task_data;
    task_data = p_arg;
    int n = OSTCBCur->OSTCBId;
    OSTCBCur->OSTCBPeriodic = task_data->TaskPeriodic;


    while (1)
    {
        OSTCBCur->OSTCBStartime = OSTimeGet();
        while (1) {
            if (OSTimeGet() >= OSTCBCur->OSTCBPeriodic * (OSTCBCur->OSTCBJobNum + 1) + OSTCBCur->OSTCBArriveTime) {
                OS_Sched();
            }
            if (OSTCBCur->OSTCBWorkTime == OSTCBCur->OSTCBExecutionTime) {

                OSTimeDly(OSTCBCur->OSTCBPeriodic * (OSTCBCur->OSTCBJobNum + 1) + OSTCBCur->OSTCBArriveTime - OSTimeGet(
                break;
            }
        }
    }
```

在main裡面task的內容部分，當任務時間大於等於Deadline的時間時，會進行OS_Sched()，再從 OS_Sched() 進到 OS_SchedNew() 裡面，另外，若任務使用CPU的時間等於執行時間的話，就將任務delay到下一次週期到來的時間。