

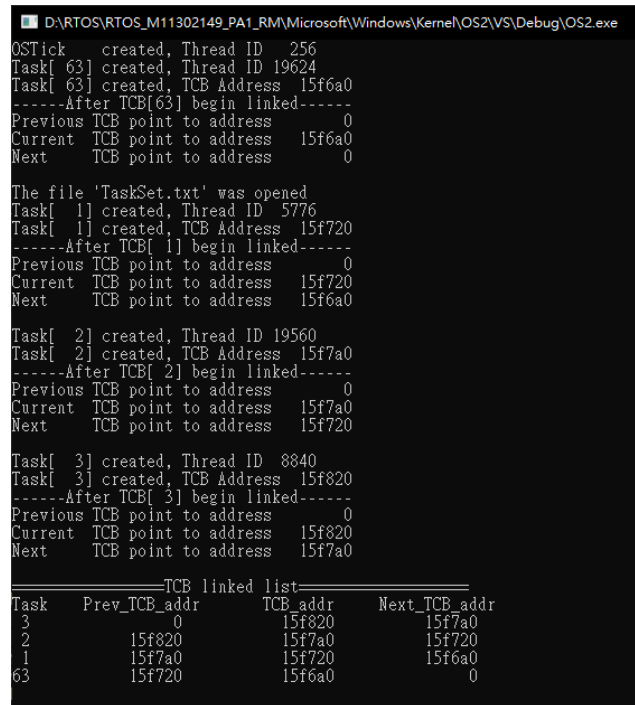
# RTOS PA #1 Report

Name: 趙孟哲

Student ID : M11302149

## [ PART I ] Task Control Block Linked List

### A. The screenshot results. (10%)



```
D:\RTOS\RTOS_M11302149_PA1_RM\Microsoft\Windows\Kernel\OS2\VS\Debug\OS2.exe
OSTick created, Thread ID 256
Task[ 63] created, Thread ID 19624
Task[ 63] created, TCB Address 15f6a0
-----After TCB[63] begin linked-----
Previous TCB point to address 0
Current TCB point to address 15f6a0
Next TCB point to address 0

The file 'TaskSet.txt' was opened
Task[ 1] created, Thread ID 5776
Task[ 1] created, TCB Address 15f720
-----After TCB[ 1] begin linked-----
Previous TCB point to address 0
Current TCB point to address 15f720
Next TCB point to address 15f6a0

Task[ 2] created, Thread ID 19560
Task[ 2] created, TCB Address 15f7a0
-----After TCB[ 2] begin linked-----
Previous TCB point to address 0
Current TCB point to address 15f7a0
Next TCB point to address 15f720

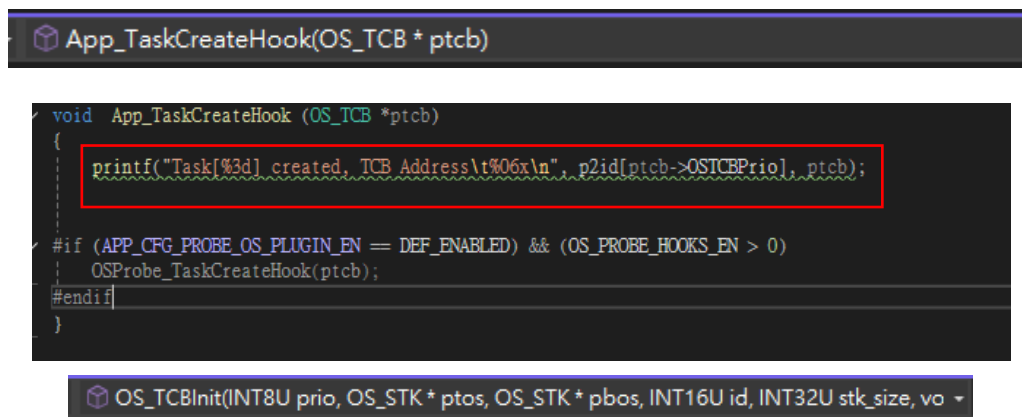
Task[ 3] created, Thread ID 8840
Task[ 3] created, TCB Address 15f820
-----After TCB[ 3] begin linked-----
Previous TCB point to address 0
Current TCB point to address 15f820
Next TCB point to address 15f7a0

-----TCB linked list-----
Task  Prev_TCB_addr  TCB_addr  Next_TCB_addr
3      0             15f820    15f7a0
2     15f820        15f7a0    15f720
1     15f7a0        15f720    15f6a0
63    15f720        15f6a0     0
```

### B. A report that describes your implementation(10%).

本部分主要在  $\mu\text{C}/\text{OS-II}$  核心中加入額外輸出，用以觀察 TCB (Task Control Block) 以及其 linked list 之狀態。

我所修改的程式位於，app\_hook.c 的 App\_taskHook() 中當 task 創建時印出 TCB 位址。



```
App_TaskCreateHook(OS_TCB *ptcb)

void App_TaskCreateHook (OS_TCB *ptcb)
{
    printf("Task[%3d] created, TCB Address\t%06x\n", p2id[ptcb->OSTCBPrio], ptcb);

    #if (APP_CFG_PROBE_OS_PLUGIN_EN == DEF_ENABLED) && (OS_PROBE_HOOKS_EN > 0)
        OSProbe_TaskCreateHook(ptcb);
    #endif
}
```

```
OS_TCBInit(INT8U prio, OS_STK *ptos, OS_STK *pbos, INT16U id, INT32U stk_size, vo
```

在 os\_core.c 的 OSTCBInit() 最後，於插入 TCB 至 OSTCBLIST 時，額外輸出目前 TCB 的位址，以及前後指向的位址

```
printf("-----After TCB[%2d] begin linked-----\n", p2id[ptcb->OSTCBPrio]);
printf("Previous TCB point to address\t%6x\n", (unsigned int)(ptcb->OSTCBPrev));
printf("Current TCB point to address\t%6x\n", (unsigned int)(ptcb));
printf("Next TCB point to address\t%6x\n\n", (unsigned int)(ptcb->OSTCBNext));
return (OS_ERR_NONE);
}
OS_EXIT_CRITICAL();
/* (OS_ERR_TASK_NO_MORE_TCB) */
```

在 main() 函數中，根據讀檔的結果創建可變數量的 task，OS\_Start() 前印出所有 TCB 狀況。

```
main(void)

/* Creat Task Set */
for (int i=0 ;i<TASK_NUMBER; i++)
{
    OSTaskCreateExt(task,
        &TaskParameter[i],
        &Task_STK[i][TASK_STACKSIZE - 1],
        TaskParameter[i].TaskPriority,
        TaskParameter[i].TaskID,
        &Task_STK[i][0],
        TASK_STACKSIZE,
        &TaskParameter[i],
        (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));
}

/*Create Task Set*/
TraverseTCBLIST();
if ((Output_err = fopen_s(&Output_fp, "../Output.txt", "a")) != 0) {
    printf("Error open Output.txt!\n");
}
fprintf(Output_fp, "Tick\tEvent\t\tCurrentTask_ID\tNextTaskID\tResponseTime\tPreemptionTime\tOSTimeDly\n");
printf("\nTick\tEvent\t\tCurrentTask_ID\tNextTaskID\tResponseTime\tOSTimeDly\n");
fclose(Output_fp);

OSTimeSet(0);
OSStart(); /* Start multitasking (i.e. give control to uC/OS-II) */
```

```
/*go thru TCB*/
void TraverseTCBLIST(void) {
    OS_TCB* ptcb = OSTCBLIST;
    printf("=====TCB linked list=====\\n");
    printf("Task\tPrev_TCB_addr\t\tTCB_addr\t\tNext_TCB_addr\\n");

    while (ptcb != (OS_TCB*)0) {
        printf("%2d\t\t\t%6x\t\t\t%6x\t\t\t%6x\\n",
            p2id[ptcb->OSTCBPrio],
            (unsigned int)ptcb->OSTCBPrev,
            (unsigned int)ptcb,
            (unsigned int)ptcb->OSTCBNext);
        ptcb = ptcb->OSTCBNext;
    }
}
/*go thru TCB*/
```

可以知道：OSTCB 指向最後一個被創建的 TCB，idle 最先被創建。

## [ PART II ] RM Scheduler Implementation [70%]

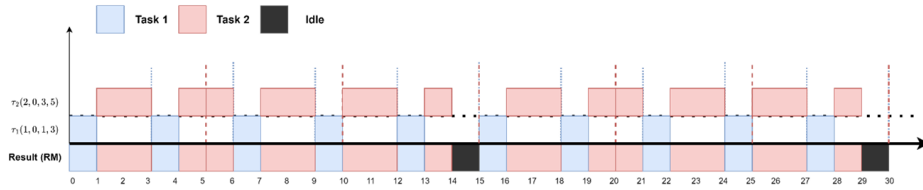
### A. Schedule results of examples. (25%)

#### Taskset I

1 0 1 3

2 0 3 5

| Tick | Event      | CurrentTask ID | NextTaskID   | ResponseTime | PreemptionTime | OSTimeDly |
|------|------------|----------------|--------------|--------------|----------------|-----------|
| 1    | Completion | task( 1)( 0)   | task( 2)( 0) | 1            | 0              | 2         |
| 3    | Preemption | task( 2)( 0)   | task( 1)( 1) |              |                |           |
| 4    | Completion | task( 1)( 1)   | task( 2)( 0) | 1            | 0              | 2         |
| 5    | Completion | task( 2)( 0)   | task( 2)( 1) | 5            | 2              | 0         |
| 6    | Preemption | task( 2)( 1)   | task( 1)( 2) |              |                |           |
| 7    | Completion | task( 1)( 2)   | task( 2)( 1) | 1            | 0              | 2         |
| 9    | Completion | task( 2)( 1)   | task( 1)( 3) | 4            | 1              | 1         |
| 10   | Completion | task( 1)( 3)   | task( 2)( 2) | 1            | 0              | 2         |
| 12   | Preemption | task( 2)( 2)   | task( 1)( 4) |              |                |           |
| 13   | Completion | task( 1)( 4)   | task( 2)( 2) | 1            | 0              | 2         |
| 14   | Completion | task( 2)( 2)   | task(63)     | 4            | 1              | 1         |
| 15   | Preemption | task(63)       | task( 1)( 5) |              |                |           |
| 16   | Completion | task( 1)( 5)   | task( 2)( 3) | 1            | 0              | 2         |
| 18   | Preemption | task( 2)( 3)   | task( 1)( 6) |              |                |           |
| 19   | Completion | task( 1)( 6)   | task( 2)( 3) | 1            | 0              | 2         |
| 20   | Completion | task( 2)( 3)   | task( 2)( 4) | 5            | 2              | 0         |
| 21   | Preemption | task( 2)( 4)   | task( 1)( 7) |              |                |           |
| 22   | Completion | task( 1)( 7)   | task( 2)( 4) | 1            | 0              | 2         |
| 24   | Completion | task( 2)( 4)   | task( 1)( 8) | 4            | 1              | 1         |
| 25   | Completion | task( 1)( 8)   | task( 2)( 5) | 1            | 0              | 2         |
| 27   | Preemption | task( 2)( 5)   | task( 1)( 9) |              |                |           |
| 28   | Completion | task( 1)( 9)   | task( 2)( 5) | 1            | 0              | 2         |
| 29   | Completion | task( 2)( 5)   | task(63)     | 4            | 1              | 1         |
| 30   | Preemption | task(63)       | task( 1)(10) |              |                |           |



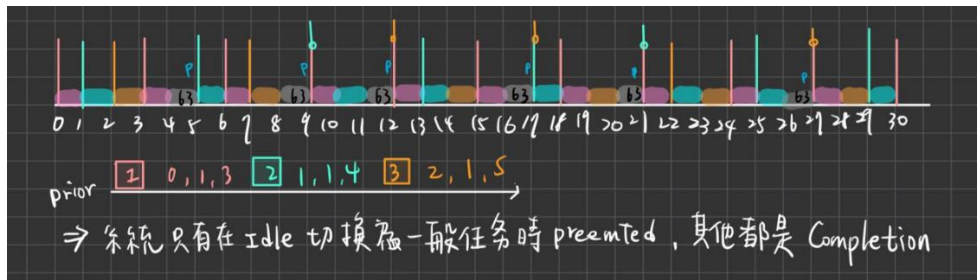
#### Taskset II

1 0 1 3

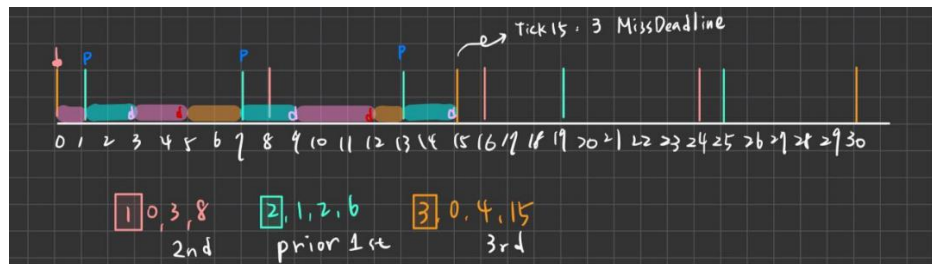
2 1 1 4

3 2 1 5

| Tick | Event      | CurrentTask ID | NextTaskID   | ResponseTime | PreemptionTime | OSTimeDly |
|------|------------|----------------|--------------|--------------|----------------|-----------|
| 1    | Completion | task( 1)( 0)   | task( 2)( 0) | 1            | 0              | 2         |
| 2    | Completion | task( 2)( 0)   | task( 3)( 0) | 1            | 0              | 3         |
| 3    | Completion | task( 3)( 0)   | task( 1)( 1) | 1            | 0              | 4         |
| 4    | Completion | task( 1)( 1)   | task(63)     | 1            | 0              | 2         |
| 5    | Preemption | task(63)       | task( 2)( 1) |              |                |           |
| 6    | Completion | task( 2)( 1)   | task( 1)( 2) | 1            | 0              | 3         |
| 7    | Completion | task( 1)( 2)   | task( 3)( 1) | 1            | 0              | 2         |
| 8    | Completion | task( 3)( 1)   | task(63)     | 1            | 0              | 4         |
| 9    | Preemption | task(63)       | task( 1)( 3) |              |                |           |
| 10   | Completion | task( 1)( 3)   | task( 2)( 2) | 1            | 0              | 2         |
| 11   | Completion | task( 2)( 2)   | task(63)     | 2            | 1              | 2         |
| 12   | Preemption | task(63)       | task( 1)( 4) |              |                |           |
| 13   | Completion | task( 1)( 4)   | task( 2)( 3) | 1            | 0              | 2         |
| 14   | Completion | task( 2)( 3)   | task( 3)( 2) | 1            | 0              | 3         |
| 15   | Completion | task( 3)( 2)   | task( 1)( 5) | 3            | 2              | 2         |
| 16   | Completion | task( 1)( 5)   | task(63)     | 1            | 0              | 2         |
| 17   | Preemption | task(63)       | task( 2)( 4) |              |                |           |
| 18   | Completion | task( 2)( 4)   | task( 1)( 6) | 1            | 0              | 3         |
| 19   | Completion | task( 1)( 6)   | task( 3)( 3) | 1            | 0              | 2         |
| 20   | Completion | task( 3)( 3)   | task(63)     | 3            | 2              | 2         |
| 21   | Preemption | task(63)       | task( 1)( 7) |              |                |           |
| 22   | Completion | task( 1)( 7)   | task( 2)( 5) | 1            | 0              | 2         |
| 23   | Completion | task( 2)( 5)   | task( 3)( 4) | 2            | 1              | 2         |
| 24   | Completion | task( 3)( 4)   | task( 1)( 8) | 2            | 1              | 3         |
| 25   | Completion | task( 1)( 8)   | task( 2)( 6) | 1            | 0              | 2         |
| 26   | Completion | task( 2)( 6)   | task(63)     | 1            | 0              | 3         |
| 27   | Preemption | task(63)       | task( 1)( 9) |              |                |           |
| 28   | Completion | task( 1)( 9)   | task( 3)( 5) | 1            | 0              | 2         |
| 29   | Completion | task( 3)( 5)   | task( 2)( 7) | 2            | 1              | 3         |
| 30   | Completion | task( 2)( 7)   | task( 1)(10) | 1            | 0              | 3         |



| Tick | Event        | CurrentTask ID | NextTaskID   | ResponseTime | PreemptionTime | OSTimeDly |
|------|--------------|----------------|--------------|--------------|----------------|-----------|
| 1    | Preemption   | task( 1)( 0)   | task( 2)( 0) | 2            | 0              | 4         |
| 3    | Completion   | task( 2)( 0)   | task( 1)( 0) | 5            | 2              | 3         |
| 5    | Completion   | task( 1)( 0)   | task( 3)( 0) | 2            | 0              | 4         |
| 7    | Preemption   | task( 3)( 0)   | task( 2)( 1) | 4            | 1              | 4         |
| 9    | Completion   | task( 2)( 1)   | task( 1)( 1) | 2            | 0              | 4         |
| 12   | Completion   | task( 1)( 1)   | task( 3)( 0) | 2            | 0              | 4         |
| 13   | Preemption   | task( 3)( 0)   | task( 2)( 2) | 2            | 0              | 4         |
| 15   | Completion   | task( 2)( 2)   | task( 3)( 0) | 2            | 0              | 4         |
| 15   | MissDeadline | task( 3)( 0)   |              |              |                |           |



## B. Implementation(40%)

首先印出標題列

```
main(void)

/* Create Task Set */
for (int i = 0; i < TASK_NUMBER; i++)
{
    OSTaskCreateExt(task,
        &TaskParameter[i],
        &Task_STK[i][TASK_STACKSIZE - 1],
        TaskParameter[i].TaskPriority,
        TaskParameter[i].TaskID,
        &Task_STK[i][0],
        TASK_STACKSIZE,
        &TaskParameter[i],
        (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));
}

/* Create Task Set */

TraverseTCBList();
if ((Output_err = fopen_s(&Output_fp, ".\\Output.txt", "a")) != 0) {
    printf("Error open Output.txt!\n");
}

fprintf(Output_fp, "Tick\\tEvent\\t\\tCurrentTask ID\\tNextTaskID\\tResponseTime\\tPreemptionTime\\tOSTimeDly\\n");
printf("\\n\\nTick\\tEvent\\t\\tCurrentTask ID\\tNextTaskID\\tResponseTime\\tOSTimeDly\\n");
fclose(Output_fp);

OSTimeSet(0);
OSStart();

/* Start multitasking (i.e. give control to uC/OS-II) */
```

在 ucosii.h 的 OS\_TCB 中，每個 TCB 新增 task 執行狀況的欄位。

```
typedef struct os_tcb {
    OS_STK      *OSTCBStkPtr;

    #if OS_TASK_CREATE_EXT_EN > 0u
        void      *OSTCBExtPtr;
        OS_STK      *OSTCBStkBottom;
        INT32U      OSTCBStkSize;
        INT16U      OSTCBOpt;
        INT16U      OSTCBId;

        INT32U remaining;

        INT32U period;
        INT32U execution_time;
        INT32U TaskNumber;

        INT32U ArriveTime;
        INT32U deadline;
        INT8U state;
        INT8U TaskID;
    #endif
}
```

OS\_TCBInit 初始化 TCB

```
OS_TCBInit(INT8U prio, OS_STK *ptos, OS_STK *pbos, INT16U id, INT32U stk_size, vo

    #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 */

        task_para_set* task = &TaskParameter[p2id[prio]-1];

        ptcb->remaining = 0;
        ptcb->period = task->TaskPeriodic;
        ptcb->execution_time = task->TaskExecutionTime;

        ptcb->ArriveTime = task->TaskArriveTime;
        ptcb->deadline = task->TaskArriveTime + task->TaskPeriodic ;
        ptcb->state = 0; // created, not arrival
        ptcb->TaskNumber = 0;
    #endif
}
```

OsStart() 要設定 tick=0 時的狀態：除了 arrival 的 task 之外其他設置 OSTCBDly，將 task 從 ready table 中移除(因為是設 task 創建時會設為 ready)。

```
OSStart(void)
void OSStart (void)
{
    OSTCBCur = OSTCBList;
    OS_TCB* ptcb = OSTCBList;
    while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) { /* Go through all TCBs in TCB list */
        OS_ENTER_CRITICAL();
        if (OSTime == ptcb->ArriveTime)
        {
            ptcb->remaining = ptcb->execution_time;
            ptcb->deadline = ptcb->period;
            ptcb->old_ArriveTime = 0;
            ptcb->state = 1;
        }
        else
        {
            INT8U y = ptcb->OSTCBY; /* Delay current task */
            OSRdyTbl[y] &= (OS_PRIO)-ptcb->OSTCBBitX;
            if (OSRdyTbl[y] == 0u) {
                OSRdyGrp &= (OS_PRIO)-ptcb->OSTCBBitY;
            }
            ptcb->OSTCBDly = ptcb->ArriveTime; /* Load ticks in TCB */
            ptcb->state = 0;
        }

        OS_EXIT_CRITICAL();
        ptcb = ptcb->OSTCBNext; /* Point at next TCB in TCB list */
    }
}
```

呼叫 `sched_new()` 選出 `highRdy` 的 task，設定執行的參數，例如執行時間、任務狀態和 `DeadLine`，然後呼叫 `OSStartHighRdy` 開始執行 `highRdy` task。

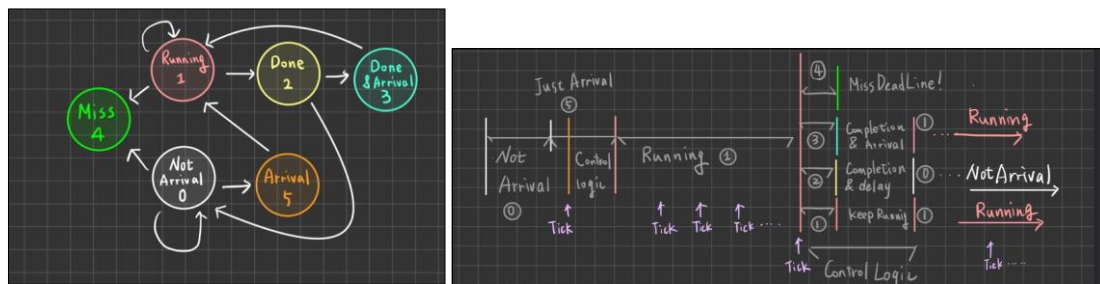
```
if (OSRunning == OS_FALSE) {
    OS_SchedNew(); /* Find highest priority's task priority number

    OSPrioCur = OSPrioHighRdy;
    OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy]; /* Point to highest priority task ready to run
    OSTCBCur = OSTCBHighRdy;

    if (OSPrioHighRdy != OS_TASK_IDLE_PRIO)
    {
        OSTCBCur->deadline = OSTCBCur->period;
        OSTCBCur->remaining = OSTCBCur->execution_time;
        OSTCBCur->state = 1;
    }

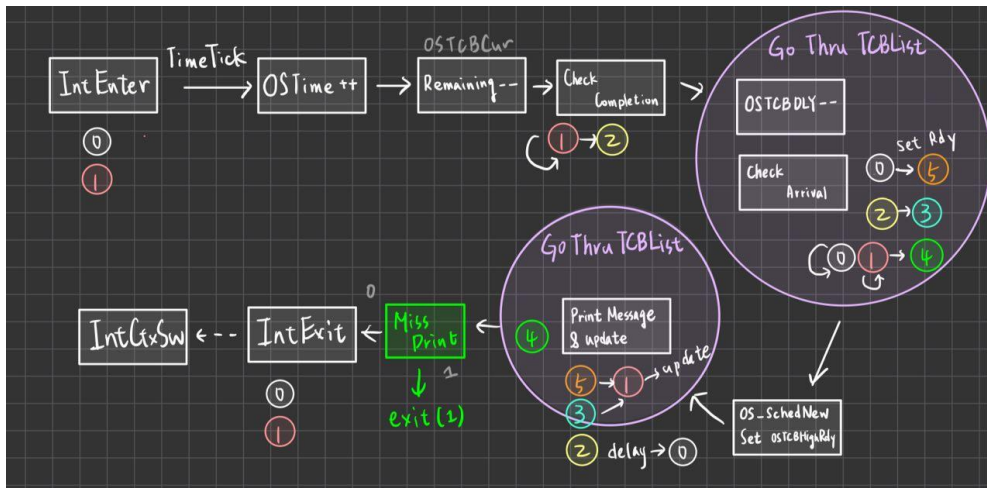
    OSStartHighRdy(); /* Execute target specific code to start task
}
```

我使用 FSM 監控 task 的執行狀態。



- **State 0: Not Arrival**  
初始狀態，任務尚未抵達系統。當系統時間到達任務指定的 `arrival time` 時，會進入狀態 5。
- **State 1: Running**  
任務被排程並執行中。根據 `remaining` 時間與 `OSTime` 判斷是否完成或超時。
- **State 2: Done**  
任務在執行完後尚未到達 `deadline`，會進入延遲期（設定 `OSTCBDly`），待重新就緒。
- **State 3: Done & Arrival**  
任務在完成的同時，也已經到了下一週期的 `arrival time`，直接產生下一個 `job` 並切換至 `Running`。
- **State 4: Miss Deadline**  
任務超過 `deadline` 仍未完成執行，進入錯誤狀態，並在訊息輸出後由系統終止模擬。
- **State 5: Just Arrival**  
任務剛抵達系統，在該 `tick` 被視為 `ready` 並加入 `ready table`，轉移至執行狀態 1。

以下是 OSTimetick 的運作流程圖。



依照 RM 排程規則，以任務 Period 為依據決定優先權，週期越短優先權越高。我在 OSTimeTick() 實作 task 運作邏輯。首先將目前執行的任務(idle 忽略)的 remaining 遞減。如果變為 0，檢查是否到下次週期，區分狀態。一開始進入中斷時會執行 IntEnter，接著 OSTime++，表示時間前進一個 tick。然後針對目前正在執行的任務 OSTCBCur 檢查是否完成執行，如果 remaining 時間為 0，就根據是否剛好達到 deadline 來設定任務狀態，若有提早完成則進入 state 2（完成但尚未到達下一週期），設定倒下次週期的 Delay 時間；若剛好達到 deadline 則進入 state 3（完成並且可以產生下一個 task）。

```

OSTimeTick(void)
{
    // Decrement remaining time for running task
    if (OSTCBCur->OSTCBPrio != OS_TASK_IDLE_PRIO)
    {
        if (OSTCBCur->remaining > 0)
        {
            OSTCBCur->remaining--;
            if (OSTCBCur->remaining == 0)
            {
                if (OSTime == OSTCBCur->deadline) {
                    OSTCBCur->state = 3;
                }
                else {
                    OSTCBCur->state = 2;
                    OSTCBCur->OSTCDBdy = OSTCBCur->deadline - OSTime;
                    OSRdyTbl[OSTCBCur->OSTCBY] &= ~(OSTCBCur->OSTCBBitX);
                    if (OSRdyTbl[OSTCBCur->OSTCBY] == 0)
                        OSRdyGrp &= ~(OSTCBCur->OSTCBBitY);
                }
            }
        }
    }
}

```

接著會走訪所有任務的 TCB linked list。每個任務會（OSTCBDly--）並檢查是否到達 arrival time。若剛好到達，state 會從 0 轉為 5。state 5 表示任務剛抵達，會在這個 tick 中被排程。同時檢查是否有任務已經超過 deadline，標記為 miss(state 4)。

```
ptcb = OSTCBList; /* Point at first TCB in TCB list */
while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) { /* Go through all TCBs in TCB list */

    OS_ENTER_CRITICAL();

    //printf("task: %2d remaining : %2d\n", ptcb->TaskID, ptcb->remaining);
    // Deadline miss
    if (ptcb->state == 1 && ptcb->remaining > 0 && OSTime >= ptcb->deadline) {
        ptcb->state = 4;
    }

    INT8U kkk = OSTime;
    if (ptcb->OSTCBDly != 0u) { /* No, Delayed or waiting for event with TO */
        if (ptcb->state == 0)
            ptcb->OSTCBDly--; /* Decrement nbr of ticks to end of delay */
        if (ptcb->OSTCBDly == 0u && ptcb->state == 0) { /* Check for timeout */
            ptcb->state = 5;
            if ((ptcb->OSTCBStat & OS_STAT_PEND_ANY) != OS_STAT_RDY) {
                ptcb->OSTCBStat &= (INT8U)~(INT8U)OS_STAT_PEND_ANY; /* Yes, Clear status flag */
                ptcb->OSTCBStatPend = OS_STAT_PEND_TO; /* Indicate PEND timeout */
            }
            else {
                ptcb->OSTCBStatPend = OS_STAT_PEND_OK;
            }

            if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == OS_STAT_RDY) { /* Is task suspended? */
                OSRdyGrp |= ptcb->OSTCBBitY; /* No, Make ready */
                OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
                OS_TRACE_TASK_READY(ptcb);
            }
        }
    }

    ptcb = ptcb->OSTCBNext; /* Point at next TCB in TCB list */
    OS_EXIT_CRITICAL();
}
```

在遍歷完 TCB 後此時 task 都以更新完 RdyTbl 了，呼叫 OS\_SchedNew 計算當前應執行的最高優先權任務，並設定 OSTCBHighRdy。

```
// 2. Scheduler: decide the next task
OS_SchedNew();
OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
```

然後第二次走訪 TCB，這時會根據任務狀態印出相對應的 log。例如 state 2 會輸出 Completion 並等待下一次 arrival，state 3 則代表任務完成後立刻產生新 task 而進入 running，state 5 也會設定 ready。



```

// 3. Second pass: print logs and finalize state transitions
ptcb = OSTCBLst;
while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
    //OS_ENTER_CRITICAL();

    if (ptcb->TaskID > 0) {
        switch (ptcb->state) {
            case 2:
                PrintTask("Completion",NULL);
                ptcb->TaskNumber++;
                ptcb->state = 0;

                break;
            case 3:
                //printf("%d Arrive task(%d)(job %d) t3\n", OSTime, ptcb->TaskID, ptcb->TaskNumber + 1);
                PrintTask("Completion",NULL);
                ptcb->ArriveTime = OSTime;
                ptcb->rcremaining = ptcb->exccution_time;
                ptcb->TaskNumber++;
                ptcb->dcadline = OSTime + ptcb->period;
                ptcb->state = 1;
                break;
            case 4:
                //printf("%d MissDeadline task(%d)(job %d)\n", OSTime, ptcb->TaskID, ptcb->TaskNumber);
                Miss_ptcb = ptcb;
                break;
            case 5:
                //printf("%d Arrive task(%d)(job %d)t5\n", OSTime, ptcb->TaskID, ptcb->TaskNumber);
                ptcb->state = 1;
                ptcb->ArriveTime = OSTime;
                ptcb->dcadline = OSTime + ptcb->period;
                ptcb->rcremaining = ptcb->exccution_time;
                break;
            default:
                break;
        }
    }

    //OS_EXIT_CRITICAL();
    ptcb = ptcb->OSTCBNext;
}

```

如果 Miss，印出 Miss 的任務並停止系統。

如果新排出的任務與目前執行任務不同，則輸出 Preemption 訊息，並執行 IntExit 與 IntCtxSw 進行任務切換。整體流程會在一個 Tick 中完成所有任務狀態的更新與排程動作。

```

if (Miss_ptcb)
{
    PrintTask("MissDeadLine",Miss_ptcb);
    exit(1);
}

if (OSTCBCur != OSTCBHighRdy && (OSTCBCur->state == 11 || OSPrrioCur == OS_TASK_IDLE_PRIO)) {
    PrintTask("Preemption",NULL);
}

```

以下是輸出 log 用的函數，跟據引數調整輸出內容。

```
static void PrintTask(char type[11], const OS_TCB* miss) {  
  
    char idle_name[13] = "task(63)";  
    char name1[13];  
    char name2[13];  
    char* curr = name1;  
    char* next = name2;  
  
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) != 0) {  
        printf("Error open Output.txt!\n");  
    }  
  
    // Format current and next task names with TaskID and TaskNumber  
    sprintf(name1, sizeof(name1), "task(%2d)(%2d)",  
            OSTCBCur->TaskID, OSTCBCur->TaskNumber);  
    if (OSTCBCur == OSTCBHighRdy)  
        sprintf(name2, sizeof(name2), "task(%2d)(%2d)",  
                OSTCBHighRdy->TaskID, OSTCBHighRdy->TaskNumber+1);  
    else  
        sprintf(name2, sizeof(name2), "task(%2d)(%2d)",  
                OSTCBHighRdy->TaskID, OSTCBHighRdy->TaskNumber);  
  
    // If current or next task is idle, use the idle_name instead  
    if (OSTCBCur->OSTCBPrio == OS_TASK_IDLE_PRIO)  
        curr = idle_name;  
    if (OSTCBHighRdy->OSTCBPrio == OS_TASK_IDLE_PRIO)  
        next = idle_name;  
  
    if (strcmp(type, "Completion")==0)  
    {  
        printf("%2u\tCompletion\t%-12s\t%-12s\t%8u\t%8u\t%7u\n",  
            OSTime,  
            curr,  
            next,  
            OSTime - OSTCBCur->ArriveTime,  
            OSTime - OSTCBCur->ArriveTime - OSTCBCur->execution_time,  
            OSTCBCur->deadline - OSTime);  
        fprintf(Output_fp, "%2u\tCompletion\t%-12s\t%-12s\t%8u\t%8u\t%7u\n",  
            OSTime,  
            curr,  
            next,  
            OSTime - OSTCBCur->ArriveTime,  
            OSTime - OSTCBCur->ArriveTime - OSTCBCur->execution_time,  
            OSTCBCur->deadline - OSTime);  
    }  
    else if (strcmp(type, "Preemption")==0)  
    {  
        printf("%2u\tPreemption\t%-12s\t%-12s\n",  
            OSTime, curr, next);  
        fprintf(Output_fp, "%2u\tPreemption\t%-12s\t%-12s\n",  
            OSTime, curr, next);  
    }  
    else if (strcmp(type, "MissDeadLine")==0)  
    {  
        printf("%2u\tMissDeadline\ttask(%2u)(%2u)\t-----\n",  
            OSTime, miss->TaskID, miss->TaskNumber);  
        fprintf(Output_fp, "%2u\tMissDeadline\ttask(%2u)(%2u)\t-----\n",  
            OSTime, miss->TaskID, miss->TaskNumber);  
    }  
}
```

Task 本身是無窮迴圈，所以只有 tick 時才會切換(由 timetic 控制)

```
void task(void* p_arg) {  
    task_para_set* task_data = (task_para_set*)p_arg;  
    INT8U id = task_data->TaskID;  
  
    while (1) {  
        OS_Dummy();  
    }  
}
```

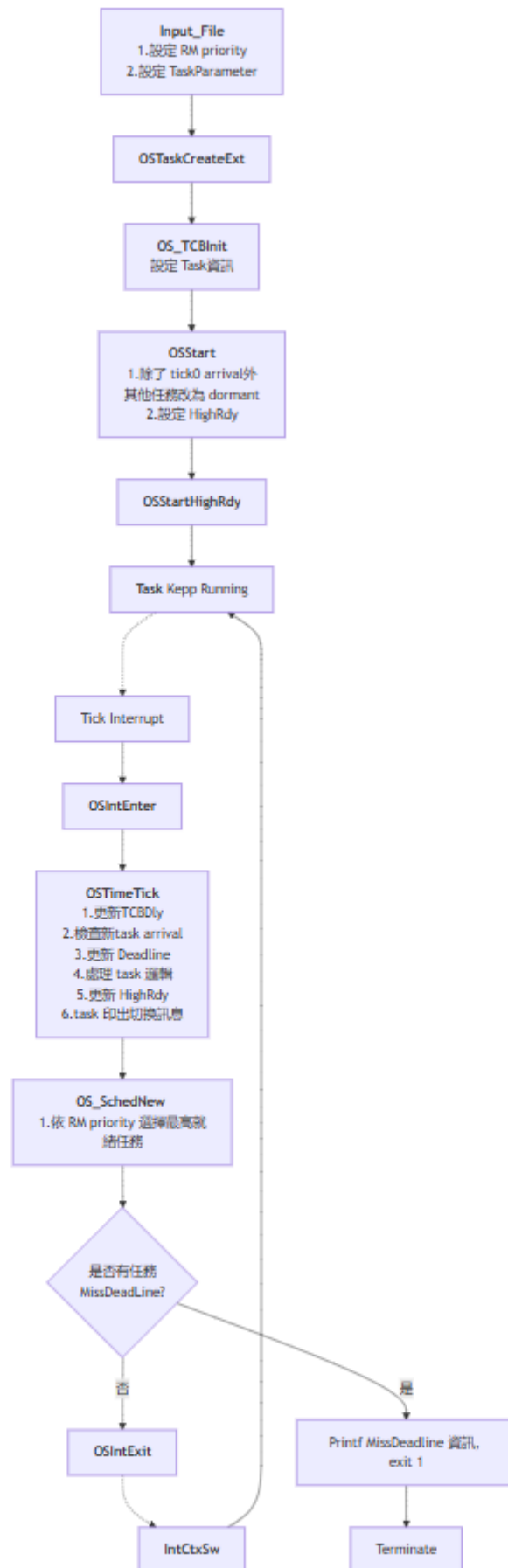
### 系統 flowchart：

下一頁是系統 flowchart。

一開始讀檔案，依據任務 period 長度設定其優先權(例如週期 5，priority 為 5。這是整個 RM 排程實作的基礎，必須在任務一創建時就確定其優先權。OSTaskCreateExt 開始建立任務，接著由 OS\_TCBInit 初始化 TCB，我在這邊加入了 RM scheduling 的控制欄位與狀態。

接下來 OSStart 是整個多工系統的啟動點，在這邊我加入了在 tick 0 arrival 以外的任務皆先 dormant 的處理，這是為了避免尚未抵達 arrival time 的任務被錯誤排程，同時也設定第一個 ready 任務作為 OSTCBHighRdy，此時第一次呼叫 OSStartHighRdy，開始執行第一個任務。

之後系統進入正常執行階段，當 timer interrupt 發生後觸發 OSTimeTick，OSTimeTick 是控制的核心。接著呼叫 OS\_SchedNew 來選中 HighRdy。然後印出 log(因為需要下次切換的任務資訊，所以要放在 OS\_SchedNew 之後)，接著 OSIntExit 離開 ISR，此時系統會視情況執行 IntCtxSw (看是否需要切換 task)。



### C. Implement and describe how to handle the deadline missing situation under RM. (5%)

系統遇到 deadline missing 的狀態會印出訊息然後停止系統，deadline missing 對於即時系統而言是致命的，應該極力避免。

```
if (Miss_ptcb)
{
    PrintTask("MissDeadLine",Miss_ptcb);
    exit(1);
}
```

## [ PART III ] FIFO Scheduler Implementation [10%]

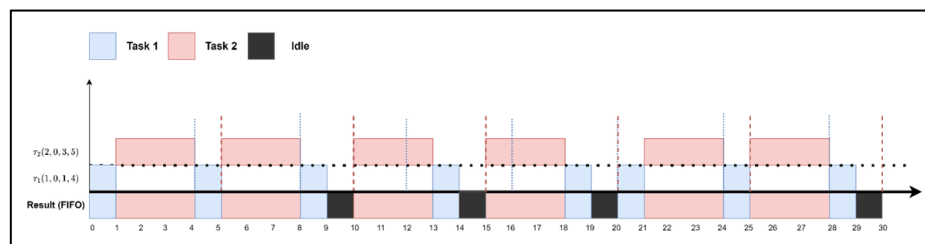
### A. schedule results of examples.

Taskset I

1 0 1 4

2 0 3 5

| Tick | Event      | CurrentTask ID | NextTaskID   | ResponseTime | PreemptionTime | OSTimeDly |
|------|------------|----------------|--------------|--------------|----------------|-----------|
| 1    | Completion | task( 1)( 0)   | task( 2)( 0) | 1            | 0              | 3         |
| 4    | Completion | task( 2)( 0)   | task( 1)( 1) | 4            | 1              | 1         |
| 5    | Completion | task( 1)( 1)   | task( 2)( 1) | 1            | 0              | 3         |
| 8    | Completion | task( 2)( 1)   | task( 1)( 2) | 3            | 0              | 2         |
| 9    | Completion | task( 1)( 2)   | task(63)     | 1            | 0              | 3         |
| 10   | Preemption | task(63)       | task( 2)( 2) |              |                |           |
| 13   | Completion | task( 2)( 2)   | task( 1)( 3) | 3            | 0              | 2         |
| 14   | Completion | task( 1)( 3)   | task(63)     | 2            | 1              | 2         |
| 15   | Preemption | task(63)       | task( 2)( 3) |              |                |           |
| 18   | Completion | task( 2)( 3)   | task( 1)( 4) | 3            | 0              | 2         |
| 19   | Completion | task( 1)( 4)   | task(63)     | 3            | 2              | 1         |
| 20   | Preemption | task(63)       | task( 1)( 5) |              |                |           |
| 21   | Completion | task( 1)( 5)   | task( 2)( 4) | 1            | 0              | 3         |
| 24   | Completion | task( 2)( 4)   | task( 1)( 6) | 4            | 1              | 1         |
| 25   | Completion | task( 1)( 6)   | task( 2)( 5) | 1            | 0              | 3         |
| 28   | Completion | task( 2)( 5)   | task( 1)( 7) | 3            | 0              | 2         |
| 29   | Completion | task( 1)( 7)   | task(63)     | 1            | 0              | 3         |
| 30   | Preemption | task(63)       | task( 2)( 6) |              |                |           |



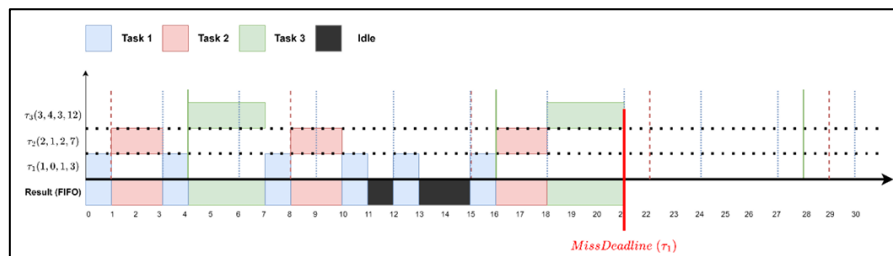
Taskset II

1 0 1 3

2 1 2 7

3 4 3 12

| Tick | Event        | CurrentTask ID | NextTaskID   | ResponseTime | PreemptionTime | OSTimeDly |
|------|--------------|----------------|--------------|--------------|----------------|-----------|
| 1    | Completion   | task( 1)( 0)   | task( 2)( 0) | 1            | 0              | 2         |
| 3    | Completion   | task( 2)( 0)   | task( 1)( 1) | 2            | 0              | 5         |
| 4    | Completion   | task( 1)( 1)   | task( 3)( 0) | 1            | 0              | 2         |
| 7    | Completion   | task( 3)( 0)   | task( 1)( 2) | 3            | 0              | 9         |
| 8    | Completion   | task( 1)( 2)   | task( 2)( 1) | 2            | 1              | 1         |
| 10   | Completion   | task( 2)( 1)   | task( 1)( 3) | 2            | 0              | 5         |
| 11   | Completion   | task( 1)( 3)   | task(63)     | 2            | 1              | 1         |
| 12   | Preemption   | task(63)       | task( 1)( 4) |              |                |           |
| 13   | Completion   | task( 1)( 4)   | task(63)     | 1            | 0              | 2         |
| 15   | Preemption   | task(63)       | task( 1)( 5) |              |                |           |
| 16   | Completion   | task( 1)( 5)   | task( 2)( 2) | 1            | 0              | 2         |
| 18   | Completion   | task( 2)( 2)   | task( 3)( 1) | 3            | 1              | 4         |
| 21   | Completion   | task( 3)( 1)   | task( 1)( 6) | 5            | 2              | 7         |
| 21   | MissDeadline | task( 1)( 6)   | -----        |              |                |           |



## B. Implement (5%)

ucosii.h: 新增 task\_node，和 task\_queue 結構去排任務。使用 Queue 去實現 FIFO。

```

/*task node queue*/
typedef struct task_node {
    OS_TCB* TaskTCB;
    INT16U TaskID;
    INT16U TaskArriveTime;
    INT16U remaining;
    INT16U TaskExecuteTime;
    INT16U TaskDeadline;
    INT16U TaskNumber;
    struct task_node* next;
} task_node;

typedef struct task_queue {
    task_node* head;
    task_node* tail;
} task_queue;
task_queue* OS_FIFO_Queue;
/*task node queue*/

```

在 main.c, main() 創建 task\_queue:

```
main(void)
```

```

OS_FIFO_Queue = (task_queue*)malloc(sizeof(task_queue));
OS_FIFO_InitQueue(OS_FIFO_Queue);

```

在 os\_core.c 的 OSStart: 如果任務 arrive 放入 Queue。

```

void OSStart (void)
{
    OSTCBCur = OSTCBList;
    OS_TCB* ptcb = OSTCBList;
    for (INT8U prio = 0; prio < OS_LOWEST_PRIO; prio++) { /* Go through all TCBs in T
        OS_TCB* ptcb = OSTCBPrioTbl[prio];
        if (ptcb == NULL) {
            continue; // Skip unused or idle task
        }
        OS_ENTER_CRITICAL();
        if (OSTime == ptcb->ArriveTime)
        {
            OS_FIFO_Enqueue(OS_FIFO_Queue, ptcb, 0);
        }
        OS_EXIT_CRITICAL();
    }
}

```

開始運作後，先宣告一個變數 `completion_flag` 表示是否有任務在這個 Tick 完成。接下來會從 FIFO 任務佇列中取出目前排在最前面的任務，這段透過 `OS_FIFO_Peek` 取得任務指標，再用 `OS_FIFO_GetCopy` 建立一份 copy 用來記錄完成狀態(因為後面可能 dequeue)。如果目前的任務不是 idle 且 `remaining` 大於零，就將 `remaining` 減 1。如果 `remaining` 減到零，表示任務執行完畢，設定 `completion_flag` 為 true，並從 FIFO 中移除該任務 (Dequeue)。

OSTimeTick(void)

```

BOOLEAN completion_flag = 0;
// Step 1: Check if the head task is done or missed deadline
task_node* current_node = (task_node*)malloc(sizeof(task_node));
task_node* current_node_ptr = OS_FIFO_Peek(OS_FIFO_Queue);
OS_FIFO_GetCopy(OS_FIFO_Queue, current_node);

if (current_node_ptr != NULL) { //null mean now is idle
    if (current_node_ptr->remaining > 0)
        current_node_ptr->remaining--;

    if (current_node_ptr->remaining == 0)
    {
        completion_flag = 1;
        OS_FIFO_Dequeue(OS_FIFO_Queue);
    }
}
}

```

接下來進行 deadline miss 檢查。因為只有要執行的 task 會在 queue 裡，所以遍歷整個 FIFO 任務 queue，只要有任務的 deadline 小於等於目前的時間，就視為錯過 deadline，將該任務的 TCB 指向 `Miss_ptcb` (假設系統抓到一個 miss 就紀錄，之後會停止)。

```

OS_TCB* Miss_ptcb = NULL;
// check miss
task_node* node = OS_FIFO_Peek(OS_FIFO_Queue);
while (node != NULL) {
    if (OSTime >= node->TaskDeadline) {
        Miss_ptcb = node->TaskTCB;
    }
    node = node->next;
}

```

然後遍歷整個 OSTCBLIST，從 prio 0 到 OS\_LOWEST\_PRIO，因為這樣 enqueue 自然會按優先權排列。如果該 prio 上有任務存在，且這個 tick 時間點符合其 arrival 條件（即 OSTime 減去任務的起始時間為其週期的整數倍），就將該任務加入 FIFO 佇列，代表產生一個新的 task。

```

for (INT8U prio = 0; prio < OS_LOWEST_PRIO; prio++) {
    OS_TCB* ptcb = OSTCBPrioTbl[prio];
    if (ptcb == NULL || prio == OS_TASK_IDLE_PRIO)
        continue;

    // Check if this is the beginning of a new job
    if ((OSTime >= ptcb->ArriveTime) &&
        ((OSTime - ptcb->ArriveTime) % ptcb->period == 0)) {

        // Enqueue the job
        //printf("task (%2u)(%2d) enqueue\n", ptcb->OSTCBId, ptcb->TaskNumber);
        OS_FIFO_Enqueue(OS_FIFO_Queue, ptcb, OSTime);
    }
}

```

當有任務完成（即 completion\_flag 為 true）或目前執行的任務是 idle 時，表示系統可能需要切換執行中的任務。此時會先檢查 FIFO 任務佇列是否為空：若 Queue 不為空，代表有其他任務等待執行，則系統會切換至佇列最前面的任務（Head\_node 對應的 TCB）；若 Queue 為空，則系統會切換至 idle task，讓系統進入閒置狀態直到下一個任務抵達。

```

task_node* Head_node = NULL;
if (!OS_FIFO_IsEmpty(OS_FIFO_Queue))
    Head_node = OS_FIFO_Peek(OS_FIFO_Queue);

if (completion_flag || OSPrioCur == OS_TASK_IDLE_PRIO)
{
    if (Head_node != NULL) {
        // There's a ready task in FIFO → run it
        OSTCBHighRdy = Head_node->TaskTCB; // Set current priority to task's ID
        OSPrioHighRdy = Head_node->TaskTCB->OSTCBPrio;
    }
    else {
        OSTCBHighRdy = OSTCBPrioTbl[OS_TASK_IDLE_PRIO];
        OSPrioHighRdy = OS_TASK_IDLE_PRIO;
    }
    // FIFO is empty → stay
}

```



接下來根據事件輸出 log。如果有任務完成，就印出 Completion 訊息，並將其 job 編號加一。如果目前是 idle 且新任務到達，就印出 Preemption 訊息。FIFO non-preemptive schedule 的 preemption 只會發生在 idle 被其他任務搶佔時。

最後再次檢查是否有任務 deadline miss，若 Miss\_ptcb 不為空，就輸出 MissDeadline 訊息並清空任務 Queue，結束模擬。

```
    if (completion_flag)
    {
        PrintTask("Completion", current_node, NULL);
        OSTCBCur->TaskNumber++;
    }
    else {
        if ((!OS_FIFO_IsEmpty(OS_FIFO_Queue)) && OSPrioCur == OS_TASK_IDLE_PRIO)
        {
            PrintTask("Preemption", Head_node, NULL);
        }
    }

    if (Miss_ptcb)
    {
        PrintTask("MissDeadLine", NULL, Miss_ptcb);
        OS_FIFO_ClearQueue(OS_FIFO_Queue);
        exit(0);
    }

    free(current_node);
}
```

以下為 Queue 相關函式(放在 app\_hook.c)：

```
void OS_FIFO_InitQueue(task_queue* q) {
    q->head = NULL;
    q->tail = NULL;
}

void OS_FIFO_Enqueue(task_queue* q, OS_TCB* task_tcb, INT32U now)
{
    task_node* new_task = (task_node*)malloc(sizeof(task_node));
    if (new_task == NULL)
        return;

    new_task->TaskTCB = task_tcb;
    new_task->TaskID = task_tcb->TaskID;
    new_task->TaskArriveTime = now;
    new_task->remaining = task_tcb->execution_time;
    new_task->TaskExecuteTime = task_tcb->execution_time;
    new_task->TaskDeadline = now + task_tcb->period;
    new_task->TaskNumber = task_tcb->TaskNumber;
    new_task->next = NULL;

    if (q->tail == NULL) {
        q->head = q->tail = new_task;
    }
    else {
        q->tail->next = new_task;
        q->tail = new_task;
    }
}
```

```

void OS_FIFO_Dequeue(task_queue* q) {
    if (q->head == NULL)
        return ;
    task_node* temp = q->head;
    q->head = q->head->next;

    if (q->head == NULL)
        q->tail = NULL;
    free(temp);
}

task_node* OS_FIFO_Peek(task_queue* q) {
    return q->head;
}

int OS_FIFO_IsEmpty(task_queue* q) {
    return q->head == NULL;
}

void OS_FIFO_ClearQueue(task_queue* q) {
    while (!OS_FIFO_IsEmpty(q)) {
        OS_FIFO_Dequeue(q);
    }
}

```

```

int OS_FIFO_GetCopy(task_queue* q, task_node* target) {
    if (q == NULL || q->head == NULL || target == NULL) return 1;

    task_node* src = q->head;

    // Copy the contents from the head node into target
    target->TaskID = src->TaskID;
    target->TaskArriveTime = src->TaskArriveTime;
    target->remaining = src->remaining;
    target->TaskDeadline = src->TaskDeadline;
    target->TaskExecuteTime = src->TaskExecuteTime;
    target->TaskNumber = src->TaskNumber;
    target->TaskTCB = src->TaskTCB;
    target->next = NULL; // Do not copy linked list pointer
    return 0;
}

```

Task\_queue 的示意圖：

