RTOS PA1

studentID: M11202158
name: 陳昱嘉 Chen, Yu-Chia

RM Scheduler Implementation

The correctness of schedule results of examples

Taskset:

1 0 1 3
2 0 3 5

Tick	Event	CurrentTaskID	NextTask ID	Response Time	Preemption Time	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)(0)	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)(3)	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:

1 0 1 3
2 1 1 4
3 2 1 5

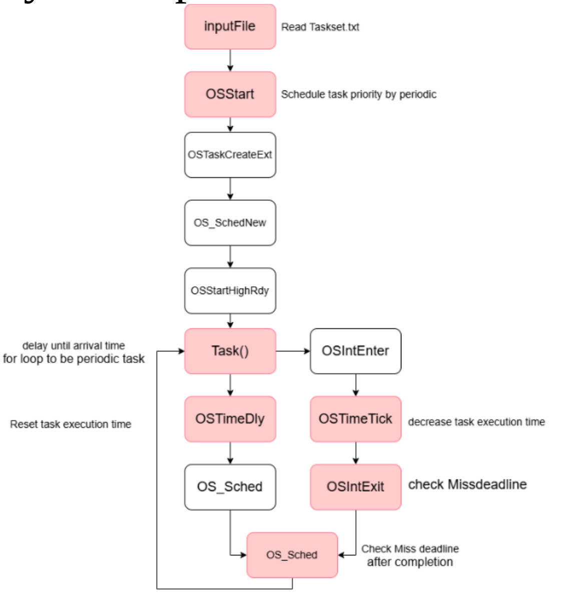
Tick	Event	CurrentTaskID	NextTask ID	Response Time	Preemption Time	OSTimeDly
1	Completion	task(1)(0)	task(2)(0)	1	0	2
2	Completion	task(2)(0)	task(3)(0)	1	0	3
3	Completion	task(3)(0)	task(1)(1)	1	0	4
4	Completion	task(1)(1)	task(63)	1	0	2
5	Preemption	task(63)	task(2)(1)			
6	Completion	task(2)(1)	task(1)(2)	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

Taskset:

1 0 3 8
2 1 2 6
3 0 4 15

Tick	Event	CurrentTaskID	NextTask ID	Response Time	Preemption Time	OSTimeDly
1	Preemption	task(1)(0)	task(2)(0)			
3	Completion	task(2)(0)	task(1)(0)	2	0	4
5	Completion	task(1)(0)	task(3)(0)	5	2	3
7	Preemption	task(3)(0)	task(2)(1)			
9	Completion	task(2)(1)	task(1)(1)	2	0	4
12	Completion	task(1)(1)	task(3)(0)	4	1	4
13	Preemption	task(3)(0)	task(2)(2)			
15	Completion	task(2)(2)	task(3)(0)	2	0	4
15	MissDeadline	task(3)(0)	-----------------			

A report that describes your implementation



Inputfile

```c
void InputFile() {
    errno_t err;
    if ((err = fopen_s(&fp, INPUT_FILE_NAME, "r")) != 0) {
        //printf("The file '%s' was opened\n", INPUT_FILE_NAME);
        printf("The file '%s' was not opened\n", INPUT_FILE_NAME);
    }

    char str[MAX];
    char* ptr;
    char* pTmp = NULL;
    int TaskInfo[INFO], i, j = 0;
    TASK_NUMBER = 0;
    while (!feof(fp)) {
        i = 0;
        memset(str, 0, sizeof(str));
        fgets(str, sizeof(str) - 1, fp);
        ptr = strtok_s(str, " ", &pTmp);
        while (ptr != NULL) {
            TaskInfo[i] = atoi(ptr);
            ptr = strtok_s(NULL, " ", &pTmp);
            if (i == 0) {
                TASK_NUMBER++;
                TaskParameter[j].TaskID = TaskInfo[i]; //task編號
            }
            else if (i == 1) {
                TaskParameter[j].TaskArriveTime = TaskInfo[i]; //到達時間
            }
            else if (i == 2) {
                TaskParameter[j].TaskExecutionTime = TaskInfo[i]; //執行時間
            }
            else if (i == 3) {
                TaskParameter[j].TaskPeriodic = TaskInfo[i]; //週期:幾個tick執行一次
            }
            i++;
        }
        TaskParameter[j].TaskNumber = 0;  //TaskNumber save the task do how many times
        TaskParameter[j].TaskPriority = j + 1;  //task priority
        TaskParameter[j].Task_need_ExecutionTime = TaskParameter[j].TaskExecutionTime;
        j++;
    }
    fclose(fp);
```

Add the ability to read the Taskset's ID, arrival time, execution time, periodicity, and Task_need_ExecutionTime, which represents the number of ticks the task still needs to complete in the current period. A task with a TaskNumber of 0 indicates the number of times the task has been completed.

```c
typedef struct task_para_set {
    INT16U TaskID;
    INT16U TaskArriveTime;
    INT16U Task_need_ExecutionTime;
    INT16U TaskExecutionTime;
    INT16U TaskPeriodic;
    INT16U TaskNumber;
    INT16U TaskPriority;
} task_para_set;
```

Each TaskParameter stores the Taskset's ID, arrival time, execution time, periodicity, Task_need_ExecutionTime (remaining ticks for completion in the current period), and TaskNumber (times the task has been completed).OSSart

```c
void  OSStart (void)
{
    if (OSRunning == OS_FALSE) {
        //sort priority by RM sheduling
        for (int i = 0; i < TASK_NUMBER; i++) {
            for (int j = i + 1; j < TASK_NUMBER; j++) {
                if (TaskParameter[i].TaskPeriodic > TaskParameter[j].TaskPeriodic) {
                    task_para_set temp = TaskParameter[i];
                    TaskParameter[i] = TaskParameter[j];
                    TaskParameter[j] = temp;
                }
            }
        }
        for (int i = 0; i < TASK_NUMBER; i++) {
            TaskParameter[i].TaskPriority = i + 1;  // 優先級從 1 開始依序分配
        }

        OS_SchedNew();                               /* Find highest priority's task
        OSPrioCur     = OSPrioHighRdy;
        OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy]; /* Point to highest priority tas
        OSTCBCur      = OSTCBHighRdy;
        OSStartHighRdy();                            /* Execute target specific code
    }
}
```

In OSStart, tasks are prioritized based on their periodicity, with shorter periods assigned higher priority.

Task

```c
void task(void* p_arg) {
    task_para_set* task_data;
    task_data = (task_para_set*)p_arg;
    //printf("TICK %d task_data: %d\n",OSTime, task_data->TaskID);
    // 初始到達時間延遲
    if (OSTime < task_data->TaskArriveTime) {
        //printf("TICK %d task_data: %d delay %d\n", OSTime, task_data->TaskID , task_data->TaskArrive
        OSTimeDly(task_data->TaskArriveTime - OSTime);
    }

    INT32U next_period = 0;
    while (1) {
        next_period = task_data->TaskPeriodic * (task_data->TaskNumber+1) + task_data->TaskArriveTime;
        //printf("delay: %d\n", task_data->TaskPeriodic - time_since_start - delay_start);
        if (task_data->Task_need_ExecutionTime==0) {
            if (next_period - OSTime > 0u) {
                OSTimeDly(next_period - OSTime);
            }
            else {
                OSTaskSwHook();
            }
        }

    }
}
```

Each task starts after an arrival time delay. In the while loop, next_period marks the next period's start. After each TimeTick, Task_need_ExecutionTime is decremented. If the task completes early, it delays until next_period - OSTime. If it finishes exactly at the next period, OSTaskSwHOOK increments TaskNumber to record task completion, preventing unnecessary context switches.OSTimeDly

```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) {                    /* See if called with scheduler locked                */
        return;
    }
    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!                             */
        //重新分配Task_need_ExecutionTime(可删除?)
        TaskParameter[OSPrioCur - 1].Task_need_ExecutionTime= TaskParameter[OSPrioCur - 1].TaskExecutionTime;
    }
}
```

When a task enters a delay, its Task_need_ExecutionTime is reset, ensuring it needs to complete the full TaskExecutionTime in the next period.

OSTimeTick

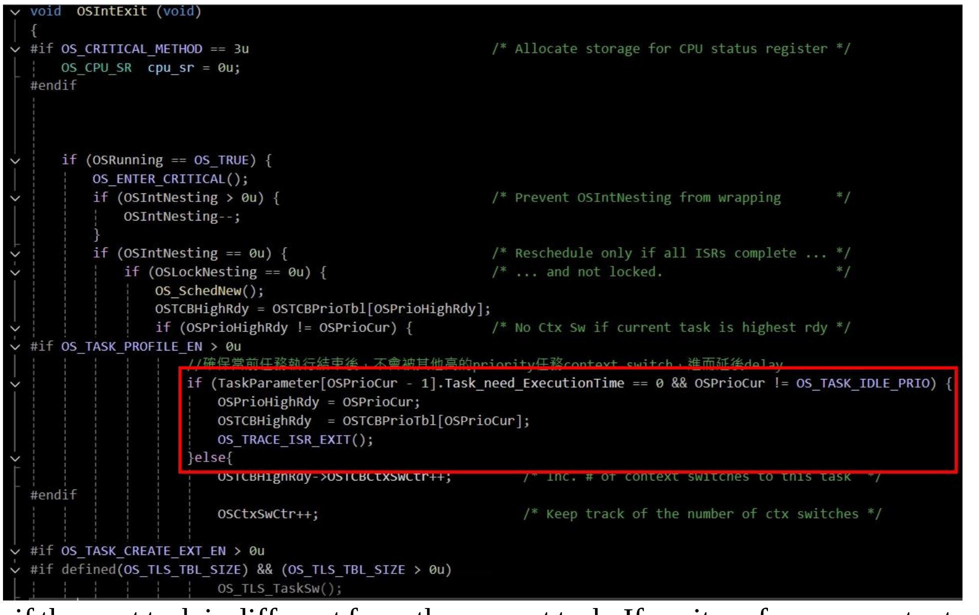
```c
void  OSTimeTick (void)
{
    OS_TCB    *ptcb;
#if OS_TICK_STEP_EN > 0u
    BOOLEAN    step;
#endif
#if OS_CRITICAL_METHOD == 3u                               /* Al
    OS_CPU_SR  cpu_sr = 0u;
#endif



#if OS_TIME_TICK_HOOK_EN > 0u
    OSTimeTickHook();                                      /* Ca
#endif
#if OS_TIME_GET_SET_EN > 0u
    OS_ENTER_CRITICAL();                                   /* Up
    OSTime++;
    if (OSPrioCur != OS_TASK_IDLE_PRIO) {
        TaskParameter[OSPrioCur - 1].Task_need_ExecutionTime--;
    }
    OS_TRACE_TICK_INCREMENT(OSTime);
    OS_EXIT_CRITICAL();
#endif
```

If the current task is not idle, its Task_need_ExecutionTime is decremented by 1, and then the task level checks whether the task is completed.

OSIntExit



```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.                      */
                OS_SchedNew();
                OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
                if (OSPrioHighRdy != OSPrioCur) {          /* No Ctx Sw if current task is highest rdy */
#if OS_TASK_PROFILE_EN > 0u
                    //確保當前任務執行結束後，不會被其他高的priority任務context switch，進而延後delay
                    if (TaskParameter[OSPrioCur - 1].Task_need_ExecutionTime == 0 && OSPrioCur != OS_TASK_IDLE_PRIO) {
                        OSPrioHighRdy = OSPrioCur;
                        OSTCBHighRdy  = OSTCBPrioTbl[OSPrioCur];
                        OS_TRACE_ISR_EXIT();
                    }else{
                        OSTCBHighRdy->OSTCBCtxSwCtr++;         /* Inc. # of context switches to this task  */
#endif
                        OSCtxSwCtr++;                          /* Keep track of the number of ctx switches */

#if OS_TASK_CREATE_EXT_EN > 0u
#if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
                        OS_TLS_TaskSw();
```

OSIntExit checks if the next task is different from the current task. If so, it performs a context switch. However, if the task just completed, it won't re-enter to finish execution, causing the current task to remain incomplete.

OS_Sched

```c
void  App_TaskSwHook (void)
{
#if (APP_CFG_PROBE_OS_PLUGIN_EN > 0) && (OS_PROBE_HOOKS_EN > 0)
    OSProbe_TaskSwHook();
#endif
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
        if (OSTime) {
            printf("%d", OSTime);
            fprintf(Output_fp, "%d", OSTime);
            if (TaskParameter[OSPrioCur - 1].Task_need_ExecutionTime == 0 && OSPrioCur != OS_TASK_IDLE_PRIO) { //如果要被切換進來的task是idle task，就print這行
                printf("\tCompletion\t");  //if the task will be done, show this line
                fprintf(Output_fp,"\tCompletion\t");
            }
            else {
                printf("\tPreemption\t");
                fprintf(Output_fp, "\tPreemption\t");
            }

            if (OSPrioCur == OS_TASK_IDLE_PRIO) { //如果要被切換出去的task是idle task，就print這行
                printf("task(%2d)\t\ttask(%2d)(%2d)", OSPrioCur, TaskParameter[OSPrioHighRdy - 1].TaskID, TaskParameter[OSPrioHighRdy - 1].TaskNumber);  //if the task being switched out is idle task, show this line
                fprintf(Output_fp, "task(%2d)\ttask(%2d)(%2d)", OSPrioCur, TaskParameter[OSPrioHighRdy - 1].TaskID, TaskParameter[OSPrioHighRdy - 1].TaskNumber);
            }
            else if (OSPrioHighRdy == OS_TASK_IDLE_PRIO) { //如果要被切換進來的task是idle task，就print這行
                printf("task(%2d)(%2d)\t\ttask(%2d)", TaskParameter[OSPrioCur - 1].TaskID, TaskParameter[OSPrioCur - 1].TaskNumber, OSPrioHighRdy);  //if the task will be switched in is idle task, show this line
                fprintf(Output_fp, "task(%2d)(%2d)\ttask(%2d)", TaskParameter[OSPrioCur - 1].TaskID, TaskParameter[OSPrioCur - 1].TaskNumber, OSPrioHighRdy);
            }
            else { //如果兩個task在做context switch，就print這行
                printf("task(%2d)(%2d)\t\ttask(%2d)(%2d)", TaskParameter[OSPrioCur - 1].TaskID, TaskParameter[OSPrioCur - 1].TaskNumber, TaskParameter[OSPrioHighRdy - 1].TaskID, TaskParameter[OSPrioHighRdy - 1].TaskNumber);  //if the
                fprintf(Output_fp, "task(%2d)(%2d)\ttask(%2d)(%2d)", TaskParameter[OSPrioCur - 1].TaskID, TaskParameter[OSPrioCur - 1].TaskNumber, TaskParameter[OSPrioHighRdy - 1].TaskID, TaskParameter[OSPrioHighRdy - 1].TaskNumber);
            }
```

OS_Sched is executed during delays or interrupt context switches. It contains OS_TASK_SW, which runs the OSTaskSwHook function. During a context switch, it calls App_TaskSwHook to check the current task's completion status using TaskParameter[OSPrioCur - 1].Task_need_ExecutionTime.

```c
            if (TaskParameter[OSPrioCur -1].Task_need_ExecutionTime == 0 && OSPrioCur != OS_TASK_IDLE_PRIO) { //done
                int response_time   = OSTime - TaskParameter[OSPrioCur - 1].TaskNumber * TaskParameter[OSPrioCur - 1].TaskPeriodic - TaskParameter[OSPrioCur - 1].TaskArriveTime;
                int preemption_time = response_time - TaskParameter[OSPrioCur - 1].TaskExecutionTime;
                int OSTimeDly       = TaskParameter[OSPrioCur - 1].TaskPeriodic - response_time;
                printf("\t%d\t\t%d\t\t%d\n", response_time, preemption_time, OSTimeDly);
                fprintf(Output_fp, "\t%d\t\t%d\t\t\t%d\n", response_time, preemption_time, OSTimeDly);

                //記錄這個task做幾次
                TaskParameter[OSPrioCur - 1].TaskNumber++;
                TaskParameter[OSPrioCur - 1].Task_need_ExecutionTime = TaskParameter[OSPrioCur - 1].TaskExecutionTime;

                //檢查Missdeadline
                for (int i = 0; i < TASK_NUMBER; i++) {
                    int response_time_i = OSTime - TaskParameter[i].TaskNumber * TaskParameter[i].TaskPeriodic - TaskParameter[i].TaskArriveTime;
                    int OSTimeDly_i = TaskParameter[i].TaskPeriodic - response_time_i;
                    if (OSTimeDly_i <= 0 && TaskParameter[OSPrioCur - 1].Task_need_ExecutionTime != 0) {

                        printf("%d\tMissDeadline\ttask( %d)( %d)\t\t----------------- \n", OSTime, TaskParameter[i].TaskID, TaskParameter[i].TaskNumber);
                        fprintf(Output_fp, "%d\tMissDeadline\ttask( %d)( %d)\t\t----------------- \n", OSTime, TaskParameter[i].TaskID, TaskParameter[i].TaskNumber);
                        OSRunning = OS_FALSE;
                        fclose(Output_fp);
                        system("pause");
                        exit(0);
                    }
                }



            }
            else {
                printf("\n");
                fprintf(Output_fp, "\n");
            }


        }
    }
    fclose(Output_fp);

}
```

When the condition TaskParameter[OSPrioCur - 1].Task_need_ExecutionTime == 0 occurs, it indicates task completion. At this point, the system displays the response time, preemption time, and OSTimeDly.

response_time = response_time = OSTime - TaskParameter[OSPrioCur - 1].TaskNumber * TaskParameter[OSPrioCur - 1].TaskPeriodic - TaskParameter[OSPrioCur - 1].TaskArriveTime;

preemption_time = response_time - TaskParameter[OSPrioCur - 1].TaskExecutionTime;

OSTimeDly = TaskParameter[OSPrioCur - 1].TaskPeriodic - response_time;

After the task completes, if the next period starts immediately, TaskNumber is incremented by 1, and Task_need_ExecutionTime is reset.

handle the deadline missing situation under RM

There are three scenarios for checking: after task completion, when a task is preempted, and when a task is currently running.

Completion > Missdeadline > Preemption

```c
if (TaskParameter[OSPrioCur -1].Task_need_ExecutionTime == 0 && OSPrioCur != OS_TASK_IDLE_PRIO) { //done
    int response_time   = OSTime - TaskParameter[OSPrioCur - 1].TaskNumber * TaskParameter[OSPrioCur - 1].TaskPeriodic - TaskParameter[OSPrioCur - 1].TaskArriveTime;
    int preemption_time = response_time - TaskParameter[OSPrioCur - 1].TaskExecutionTime;
    int OSTimeDly       = TaskParameter[OSPrioCur - 1].TaskPeriodic - response_time;
    printf("\t%d\t\t%d\t\t%d\n", response_time, preemption_time, OSTimeDly);
    fprintf(Output_fp, "\t%d\t\t%d\t\t\t%d\n", response_time, preemption_time, OSTimeDly);

    //記錄這個task做幾次
    TaskParameter[OSPrioCur - 1].TaskNumber++;
    TaskParameter[OSPrioCur - 1].Task_need_ExecutionTime = TaskParameter[OSPrioCur - 1].TaskExecutionTime;

    //檢查Missdeadline
    for (int i = 0; i < TASK_NUMBER; i++) {
        int response_time_i = OSTime - TaskParameter[i].TaskNumber * TaskParameter[i].TaskPeriodic - TaskParameter[i].TaskArriveTime;
        int OSTimeDly_i = TaskParameter[i].TaskPeriodic - response_time_i;
        if (OSTimeDly_i <= 0 && TaskParameter[OSPrioCur - 1].Task_need_ExecutionTime != 0) {

            printf("%d\tMissDeadline\ttask( %d)( %d)\t\t----------------- \n", OSTime, TaskParameter[i].TaskID, TaskParameter[i].TaskNumber);
            fprintf(Output_fp, "%d\tMissDeadline\ttask( %d)( %d)\t\t----------------- \n", OSTime, TaskParameter[i].TaskID, TaskParameter[i].TaskNumber);
            OSRunning = OS_FALSE;
            fclose(Output_fp);
            system("pause");
            exit(0);
        }
    }

}
```

In App_TaskSwHook, after a task completes, it immediately checks if any task has missed its deadline in the current tick.

The other two scenarios are checked within OSIntExit.

```c
if (TaskParameter[OSPrioCur - 1].Task_need_ExecutionTime == 0 && OSPrioCur != OS_TASK_IDLE_PRIO) {
    OSPrioHighRdy = OSPrioCur;
    OSTCBHighRdy  = OSTCBPrioTbl[OSPrioCur];
    OS_TRACE_ISR_EXIT();
}else{
    OSTCBHighRdy->OSTCBCtxSwCtr++;         /* Inc. # of context switches to this task  */

    OSCtxSwCtr++;                          /* Keep track of the number of ctx switches */

EXT_EN > 0u
TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
    OS_TLS_TaskSw();


    OS_TRACE_ISR_EXIT_TO_SCHEDULER();

    //preemption前檢查Missdeadline
    for (int i = 0; i < TASK_NUMBER; i++) {
        int response_time = OSTime - TaskParameter[i].TaskNumber * TaskParameter[i].TaskPeriodic - TaskParameter[i].TaskArriveTime;
        int OSTimeDly = TaskParameter[i].TaskPeriodic - response_time;
        if (OSTimeDly <=0) {
            if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
                printf("%d\tMissDeadline\ttask( %d)( %d)\t\t---------------- \n", OSTime, TaskParameter[i].TaskID, TaskParameter[i].TaskNumber);
                fprintf(Output_fp, "%d\tMissDeadline\ttask( %d)( %d)\t\t---------------- \n", OSTime, TaskParameter[i].TaskID, TaskParameter[i].TaskNumber);
            }
            fclose(Output_fp);
            OSRunning = OS_FALSE;
            system("pause");
            exit(0);
        }
    }

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

This occurs when a task is preempted, and before the context switch happens, it checks whether any task has missed its deadline.

```c
        } else {
            //沒有發生context switch,檢查有無missdeadline
            if (TaskParameter[OSPrioCur-1].Task_need_ExecutionTime!=0) {
                for (int i = 0; i < TASK_NUMBER; i++) {
                    int response_time = OSTime - TaskParameter[i].TaskNumber * TaskParameter[i].TaskPeriodic - TaskParameter[i].TaskArriveTime;
                    int OSTimeDly = TaskParameter[i].TaskPeriodic - response_time;
                    if (OSTimeDly <= 0) {
                        if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
                            printf("%d\tMissDeadline\ttask( %d)( %d)\t\t---------------- \n", OSTime, TaskParameter[i].TaskID, TaskParameter[i].TaskNumber);
                            fprintf(Output_fp, "%d\tMissDeadline\ttask( %d)( %d)\t\t---------------- \n", OSTime, TaskParameter[i].TaskID, TaskParameter[i].TaskNumber);
                        }
                        fclose(Output_fp);
                        OSRunning = OS_FALSE;
                        exit(0);
                    }
                }
            }
            OS_TRACE_ISR_EXIT();
        }
    } else {
        OS_TRACE_ISR_EXIT();
```

Finally, in the new tick, if the current task's TaskParameter[OSPrioCur-1].Task_need_ExecutionTime != 0, it is first checked. If Task_need_ExecutionTime == 0, it means the task has completed, so it proceeds to the task level to handle the completion before checking for missed deadlines.