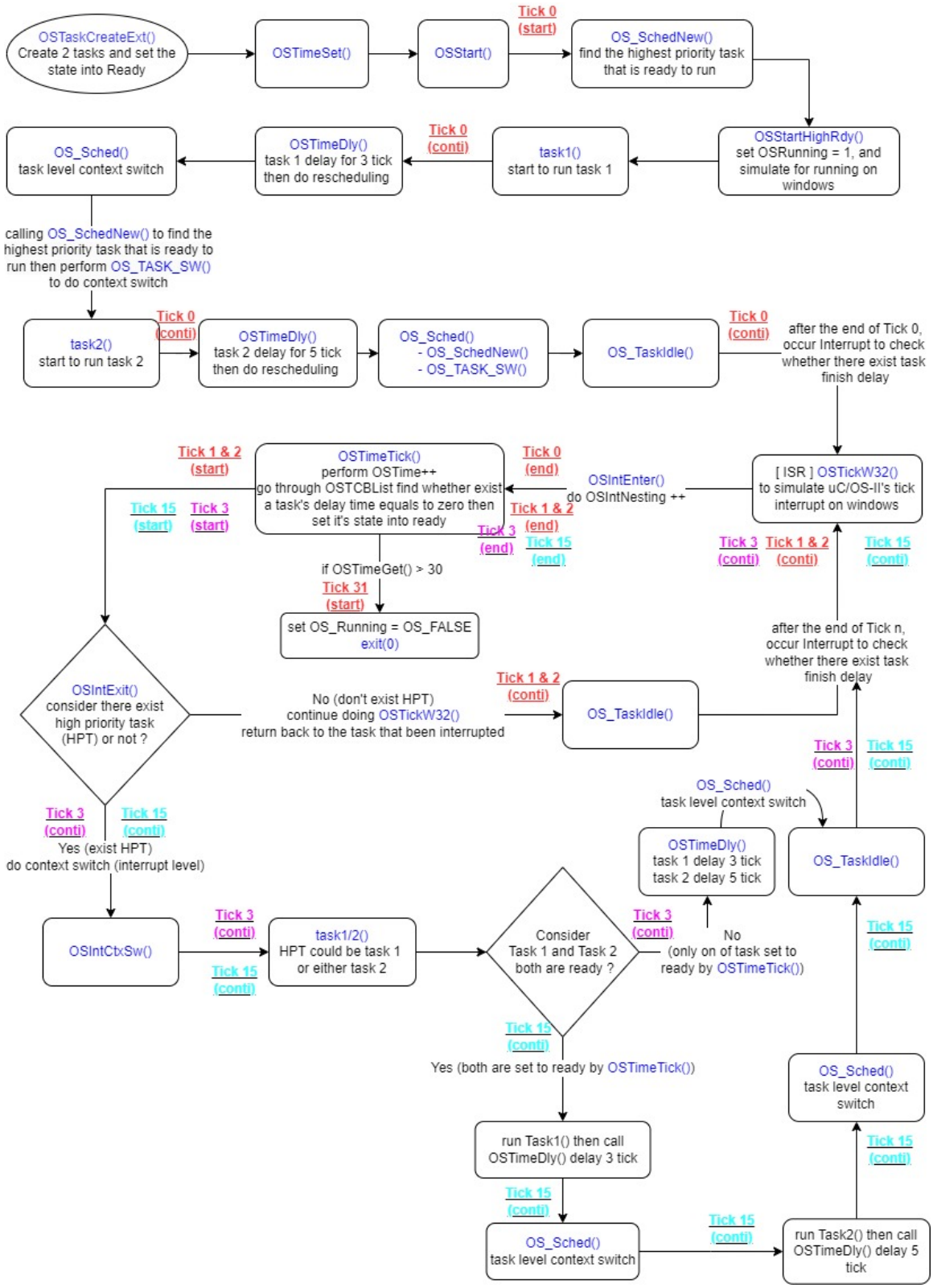
RTOS_M11102155_HW1

Name : 楊鏡業

Student ID : M11102155

(a) System flow



The sign of the tick on the system flow means the each tick will go through which process, tick will despite into three parts, start, conti, and end. start means the start of the tick, conti means the tick continue going, and end means the end of the tick. (Cause the figure of the flow might be not clear enough so i provide a .jpg in the assignment file)

Explain of each tick on the system flow :

Tick 0 :

Start when enter OSStart(), and doing task1() then delay task 1, perform task level context switch by calling OS_Sched(). After context switch start to execute task2() then delay task 2 , and task level context switch occur again, switching on OS_TaskIdle().

At the end of Tick 0 occur interrupt executing ISR OSTickW32(), to check whether exist task finish delaying. First in OSTickW32(), perform OSIntEnter() to increase OSIntNesting, then execute OSTimeTick().

In OSTimeTick(), will increase OSTime, a global variable to recording the recent system tick, after doing this the tick 0 really ended.

Tick 1 & 2 :

After increasing OSTime the tick become 1, and start to go through OSTCBList find whether exist a task's delay time equals to zero and set it's state into ready, else do nothing just decrease the delay count or set it into 1 cause some condition such that the task isn't ready althogh the delay tick become to zero.

After OSTimeTick(), doing OSIntExit() for consider exist high priority task(HPT) to perform interrupt level context switch, in this assignment sepcial case there is no other HTP found by OS, so it

will continue doing ISR OSTickW32() and pop out all thing from stack execute OS_TaskIdle() until end of Tick 1, so asTick 2.

Tick 3 :

At Tick 3 Task 1 wakes up after performing OSTimeTick(), when doing OSIntExit() find exist HPT then call OSIntCtxSw() executing interrupt level context switch, OSIntCtxSw() do less thing than task level context switch OS_TASK_SW(). Task 1 start to run on cpu and perform delay for 3 ticks, and consider whether task 2 is either awake, the case here is no, so just do task level context switch OS_Sched() switch into OS_TASKIdle() and execute it until end of Tick 3 occur interrupt then perform ISR OSTickW32().

Tick 4 :

So as Tick 1 & 2

Tick 5 & 6 :

So as Tick 3

Tick 7 & 8 :

So as Tick 1 & 2

And so on

Tick 15 :

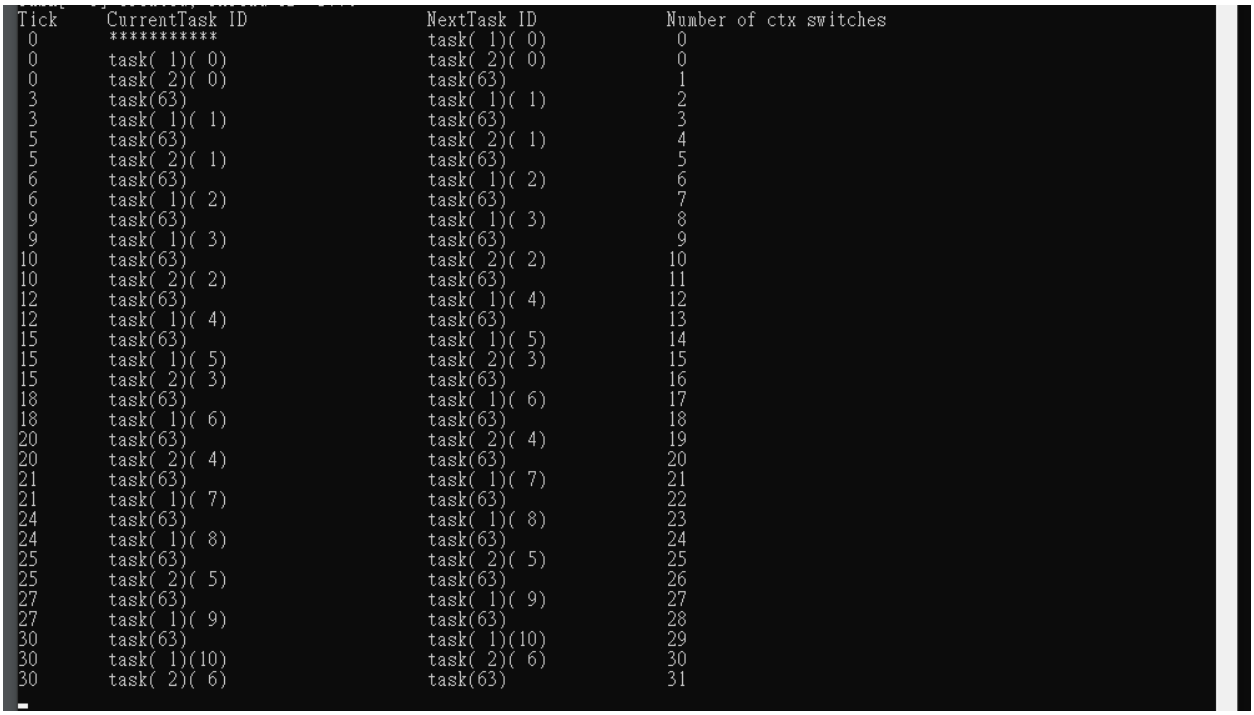
Tick 15 Task 1 and Task 2 both wake up, the front processes are same as Tick 3 until Consider both task are awake or not. the different between Tick 15 and Tick 3 is that Tick 15 has to perform a task level context switch OS_Sched once more than Tick 3.

Tick after 15 is same as Tick 1 & 2, Tick 3, and Tick 15

Tick 31 :

When performing OSTimeTick(), it'll consider whether OSTimeGet() is greater than SYSTEM_END_TIME, which we set as 30 in this assignment, if is true will set OSRunning = OS_FALSE and exit(0).

(b) Screenshot of the result



Tick	CurrentTask ID	NextTask ID	Number of ctx switches
0	**********	task(1)(0)	0
0	task(1)(0)	task(2)(0)	0
0	task(2)(0)	task(63)	1
3	task(63)	task(1)(1)	2
3	task(1)(1)	task(63)	3
5	task(63)	task(2)(1)	4
5	task(2)(1)	task(63)	5
6	task(63)	task(1)(2)	6
6	task(1)(2)	task(63)	7
9	task(63)	task(1)(3)	8
9	task(1)(3)	task(63)	9
10	task(63)	task(2)(2)	10
10	task(2)(2)	task(63)	11
12	task(63)	task(1)(4)	12
12	task(1)(4)	task(63)	13
15	task(63)	task(1)(5)	14
15	task(1)(5)	task(2)(3)	15
15	task(2)(3)	task(63)	16
18	task(63)	task(1)(6)	17
18	task(1)(6)	task(63)	18
20	task(63)	task(2)(4)	19
20	task(2)(4)	task(63)	20
21	task(63)	task(1)(7)	21
21	task(1)(7)	task(63)	22
24	task(63)	task(1)(8)	23
24	task(1)(8)	task(63)	24
25	task(63)	task(2)(5)	25
25	task(2)(5)	task(63)	26
27	task(63)	task(1)(9)	27
27	task(1)(9)	task(63)	28
30	task(63)	task(1)(10)	29
30	task(1)(10)	task(2)(6)	30
30	task(2)(6)	task(63)	31

(c) Code modify and results

The code I modify will wrapped between

```
#ifdef M11102155_HW1

    # modify code !!!

#endif /* M11102155_HW1 */
```

usos_ii.h :

for using #ifdef M11102155_HW1 #endif in all code segament.

```
#ifndef   OS_uCOS_II_H
#define   OS_uCOS_II_H

#define M11102155_HW1

#ifdef __cplusplus
extern "C" {
#endif
```

app_hooks.c :

modify the priority of task 1 and 2 into 1 and 2.

First, declar a new variable priority_count as 1.

Second, assign priority_count.

```
    char* ptr;
    char* pTmp = NULL;
    int TaskInfo[INFO], i, j = 0;

#ifdef M11102155_HW1

    int priority_count = 1;

#endif /* M11102155_HW1 */
```

```c
#ifndef M11102155_HW1

    /* Initial Priority */
    TaskParameter[j].TaskPriority = j;

#elif defined M11102155_HW1

    TaskParameter[j].TaskPriority = priority_count;
    priority_count++;

#endif /* M11102155_HW1 */

    ;;;
  }
  fclose(fp);
  /* read file */
}
```

main.c :

print the title of output for tracing.

```c
#ifdef M11102155_HW1
    printf("Tick \t CurrentTask ID \t\t NextTask ID \t\t Number of ctx switches\n");

#endif /* M11102155_HW1 */

    OSTimeSet(0);
    OSStart();                                          /* start multitasking (i.e. give control to uc/os-ii)   */

    while (DEF_ON) {                                    /* should never get here.                               */
        ;
    }
}
```

os_core.c :

OSStart() :

open the Output.txt, cause there is no running task recently, print the selected next running highest priority task (task 1, priority 1), and output the infomation to Output.txt

```c
void  OSStart (void)
{
    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    */

#ifdef M11102155_HW1
        // Open the Output file.
        if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) != 0)
        {
            printf("Error :: Output File Open Error !!! ");
            exit(0);
        }
        printf("%2d\t **********   \t\t\t task(%2d)(%2d)\t\t %2d\n", OSTime, OSTCBHighRdy-> OSTCBId, OSTCBHighRdy-> OSTCBCtxSwCtr, OSCtxSwCtr);
        fprintf(Output_fp, "%2d\t **********   \t\t\t task(%2d)(%2d)\t\t %2d\n", OSTime, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBCtxSwCtr, OSCtxSwCtr);
#endif /* M11102155_HW1 */

        OSTCBCur      = OSTCBHighRdy;
        OSStartHighRdy();                       /* Execute target specific code to start task     */       // set OSRunning = 1u
    }
}
```

OS_Sched() :

Modify task level context swtich.

First, close the increasing of two variable, local variable, the next highest priority task’s OSTCBCtxSwCtr and global variable, recording the total number of context switch OSCtxSwCtr.

Second, print the informations and output to Output.txt, if the next task is idle task then the task ID will print as the priority of the idle task and won’t show the number of context switch that happen on idle task. Then we increase the local variable that record the number of context switch of the current task will be swap out cpu, means we are recording the number of the task that been swap out cpu.

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



    OS_ENTER_CRITICAL();
    if (OSIntNesting == 0u) {                   /* Schedule only if all ISRs done and ...       */
        if (OSLockNesting == 0u) {              /* ... scheduler is not locked                  */
            OS_SchedNew();
            OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
            if (OSPrioHighRdy != OSPrioCur) {   /* No Ctx Sw if current task is highest rdy     */
#ifndef M11102155_HW1
 #if OS_TASK_PROFILE_EN > 0u
                OSTCBHighRdy->OSTCBCtxSwCtr++;  /* Inc. # of context switches to this task      */
 #endif

                OSCtxSwCtr++;                   /* Increment context switch counter             */
 #endif /* M11102155_HW1 */

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

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

#ifdef M11102155_HW1
                if (OSTCBHighRdy->OSTCBPrio == 63)
                {
                    printf("%2d\t task(%2d)(%2d)   \t\t task(%2d)    \t\t %2d\n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->OSTCBCtxSwCtr, 63, OSCtxSwCtr);
                    fprintf(Output_fp, "%2d\t task(%2d)(%2d)   \t\t task(%2d)    \t\t %2d\n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->OSTCBCtxSwCtr, 63, OSCtxSwCtr);
                }
                else
                {
                    printf("%2d\t task(%2d)(%2d)   \t\t task(%2d)(%2d)\t\t %2d\n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->OSTCBCtxSwCtr, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBCtxSwCtr, OSCtxSwCtr);
                    fprintf(Output_fp, "%2d\t task(%2d)(%2d)   \t\t task(%2d)(%2d)\t\t %2d\n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->OSTCBCtxSwCtr, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBCtxSwCtr, OSCtxSwCtr);
                }
                OSTCBCur->OSTCBCtxSwCtr++; // After print the info then do ++ on the previous task, means the task that will swap out.
                OSCtxSwCtr++;
                //printf("%2d\t task(%2d)(%2d)   \t\t\t task(%2d)(%2d)\t\t %2d\n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->OSTCBCtxSwCtr, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBCtxSwCtr, OSCtxSwCtr);
#endif /* M11102155_HW1 */

                OS_TASK_SW();                           /* Perform a context switch                    */
            }
        }
    }
    OS_EXIT_CRITICAL();
}
```

OSIntExit() :

Modify interrupt level context switch.

Fisrt, same as previous i remain close the operation on two variable.

Second, cause in this assignment is a special case, the task happen interrupt will never be other high priority task escept idle task, task 1 and task 2 perform delay immediately, i just ignore the condition for judging whether the current task is ilde task or other task, print the current task and the next task, so as previous i remain, the task ID of the idle task is the priority of the idle task. After print the infomations and output to Output.txt, cause we don't care about the number of idle task been swap out cpu, i also ignore the increaseing of local variable of idle task's OSTCBCtxSWCtr, just increase the global variable OSCtxSwCtr.

```c
                    OS_SchedNew();
                    OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
                    if (OSPrioHighRdy != OSPrioCur) {          /* No Ctx Sw if current task is highest rdy */

#ifndef M11102155_HW1
#if OS_TASK_PROFILE_EN > 0u
                        OSTCBHighRdy->OSTCBCtxSwCtr++;         /* Inc. # of context switches to this task  */
#endif
                        OSCtxSwCtr++;                          /* Keep track of the number of ctx switches */
#endif /* M11102155_HW1 */

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

#ifdef M11102155_HW1
                        // Cause the only interrupt here is time up such that idle task is interrupt by timetick
                        printf("%2d\t task(%2d)       \t\t task(%2d)(%2d)\t\t %2d\n", OSTime, 63, OSTCBHighRdy -> OSTCBId, OSTCBHighRdy -> OSTCBCtxSwCtr, OSCtxSwCtr);
                        fprintf(Output_fp, "%2d\t task(%2d)       \t\t task(%2d)(%2d)\t\t %2d\n", OSTime, 63, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBCtxSwCtr, OSCtxSwCtr);
                        OSCtxSwCtr++;
#endif /* M11102155_HW1 */

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

OSTimeTick() :

close the output file before exit().

```c
void  OSTimeTick (void)
{
    OS_TCB    *ptcb;
#if OS_TICK_STEP_EN > 0u
    BOOLEAN    step;
#endif
#if OS_CRITICAL_METHOD == 3u                          /* Allocate storage for CPU status register     */
    OS_CPU_SR  cpu_sr = 0u;
#endif



#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++;
    OS_TRACE_TICK_INCREMENT(OSTime);
    OS_EXIT_CRITICAL();
#endif
    if (OSRunning == OS_TRUE) {
        /* Setting the end time for the OS */
        if (OSTimeGet() > SYSTEM_END_TIME)
        {
#ifdef M11102155_HW1
            // Close the output file.
            fclose(Output_fp);
#endif /* M11102155_HW1 */
            OSRunning = OS_FALSE;
            exit(0);
        }
        /* Setting the end time for the OS */
#if OS_TICK_STEP_EN > 0u
        switch (OSTickStepState) {                    /* Determine whether we need to process a tick  */
            case OS_TICK_STEP_DIS:                    /* Yes, stepping is disabled                    */
                step = OS_TRUE;
                break;
```