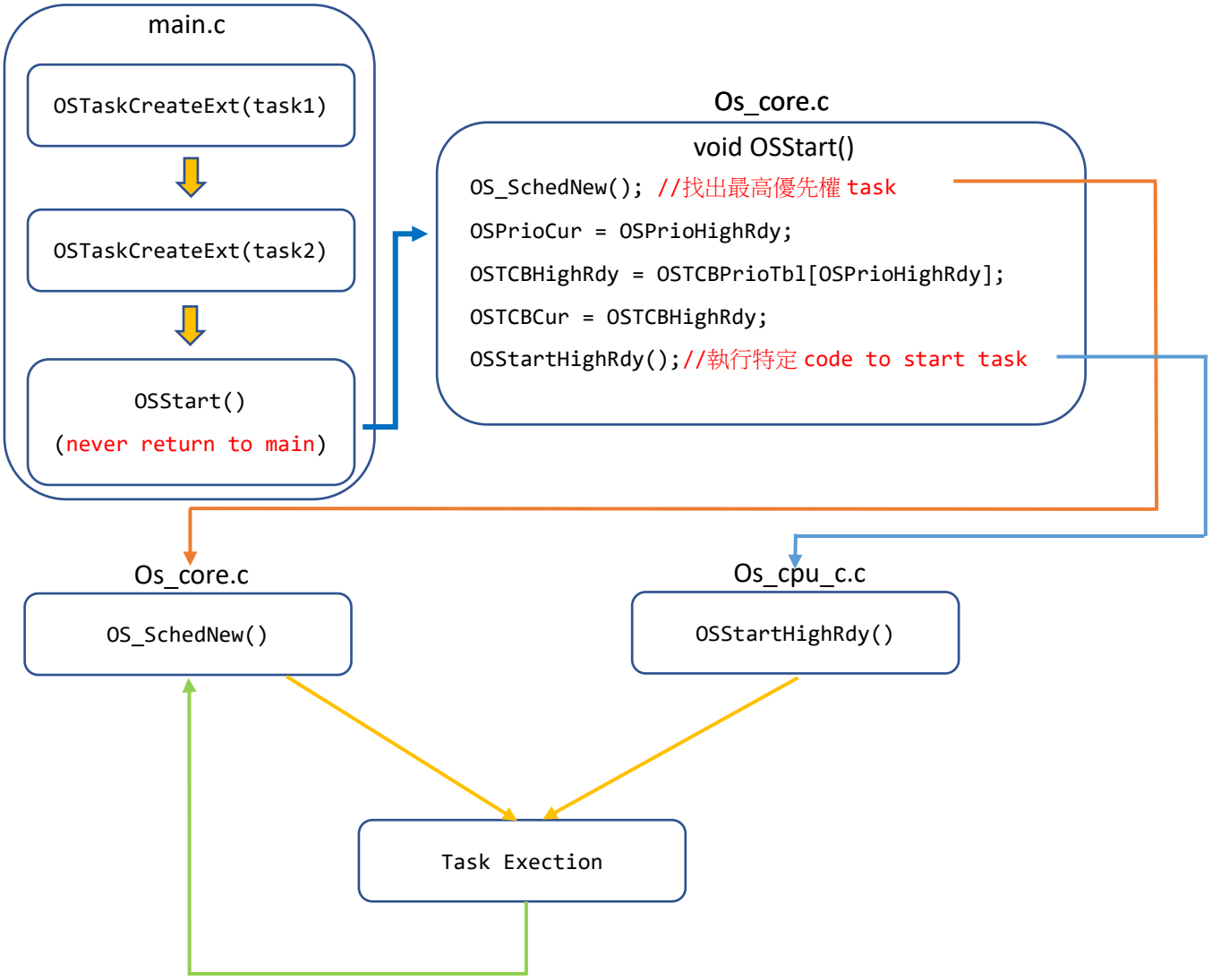
(a)

Flow Graph


main.c
OSTaskCreateExt(task1)
OSTaskCreateExt(task2)
OSStart()
(never return to main)
Os_core.c
void OSStart()
OS_SchedNew(); //找出最高優先權 task
OSPrioCur = OSPrioHighRdy;
OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
OSTCBCur = OSTCBHighRdy;
OSStartHighRdy();//執行特定 code to start task
Os_core.c
OS_SchedNew()
Os_cpu_c.c
OSStartHighRdy()
Task Exection

Function Explanation

1.OSTaskCreateExt

用來建立一個 task。參數如下：

```
INT8U  OSTaskCreateExt (void   (*task)(void *p_arg),
                        void    *p_arg,
                        OS_STK  *ptos,
                        INT8U    prio,
                        INT16U   id,
                        OS_STK  *pbos,
                        INT32U   stk_size,
                        void    *pext,
                        INT16U   opt)
{
```

參數說明:

task：一個指向 task code 的指標。

P_arg：一個指向可選擇 data 區域的指標，當 task 第一次執行時，可用來傳遞參數至 task。Task 會將 p_arg 當作傳入的參數，例如:

```
Void Task(void *p_arg){
   Infinite loop{
      /* code */
    }
}
```

Ptos：指向該 task 的 stack 頂端的指標，因此資料型態為 OS_STK

Prio：task 的優先權重。

Id：task 的 ID (範圍: 0 ~ 65535)

Pbos：指向該 task 的 stack 底部的指標，用來 check stack。

Stk_size：stack 的大小，用來 check stack，前面為 INT32U 代表包含 32-bit entries。

Pext：指向 user supplied memory location 的指標，被用來當作 TCB extension。

Opt：確認 stack checking 這個動作是否允許。

2.OSStart

此函式用來開始 multitasking，將控制權交給 os ii 進行 task management，值得注意的是，在呼叫此函式前，必須先執行 OSInit()並建立至少一個 task。

內部包含 OS_SchedNew()和 OSStartHighRdy()兩個函式。

3.OS_SchedNew

此函式會找出最高優先權的 task 並且在 ready state 中，方法為透過 bit operation。

```
y= OSUnMapTbl[OSRdyGrp];
OSPrioHighRdy = (INT8U)((y << 3u) + OSUnMapTbl[OSRdyTbl[y]]);
```

4.OSStartHighRdy

由 OSStart()呼叫，開始執行最高優先權的任務。在此函式中，必須執行以下三個指令:

(1) 呼叫 OSTaskSwHook()。

(2) 將 OSRunning 設置為 TRUE。

(3) 切換至最高優先權的 task。

```
p_stk = (OS_TASK_STK *)OSTCBHighRdy->OSTCBStkPtr; // 此為步驟(3)
```

4.OSTaskSwHook

當 task switch 發生時就會呼叫此函式，該函式可讓我們在 context switch 期間進行其他操作。值得注意的是，Interrputs 在此期間是被 disabled 的。

5.OSCtxSw(declared in os_cpu_c.c)

當有更高優先權的 task 進入 ready state，就會呼叫此函式，用來進行 task level context switch。此函式中必須執行以下幾個指令:

(1) 儲存目前 CPU 中的 Register 值。

(2) 將目前 task 的 stack pointer 儲存至該 task 的 OS_TCB 中。

(3) 呼叫 OSTaskSwHook()，因為要進行 context switch。

(4) OSTCBCur = OSTCBHighRdy，把最高優先權的 TCB 給目前的 TCB。

(5) OSPrioCur = OSPrioHighRdy，把最高優先權的 Priority 給目前的 Priority。

(6) 切換至最高優先權的任務執行。

6.OSIntCtxSw(declared in os_cpu_c.c)

此函式由 OSIntExit()呼叫，為了從 ISR 進行 context switch。此函式會執行以下幾個步驟:

(1) 呼叫 OSTaskSwHook()。

(2) OSTCBCur = OSTCBHighRdy。

(3) OSPrioCur = OSPrioHighRdy。

(4) 切換至最高優先權的任務執行。

7.OSIntExit(Declare in os_core.c)

此函式用來告知系統我們已經完成 ISR 的服務。當最後一個巢狀 ISR 完成，系統會呼叫 scheduler 確認是否有新的、更高優先權的 task 在 ready state。

Flow Explanation

首先從 main.c 開始，透過 OSTaskCreatExt()建立兩個 task。

接著進入OSStart()，開始multitasking，將控制權交給OS ii。而OSStart函式定義在os_core.c中。OSStart裡的函式OS_SschedNew()會找出最高優先權的task。進入OSStartHighRdy()，該函式定義在os_cpu_c.c，會執行特定的程式碼開始執行task，只會執行一次，並且不會回傳至OOStart。

OSStartHighRdy 會做兩件事
1.呼叫 OSTaskSwHook()，並設定 OSRunning = true(1)
2.切換至最高優先權的 task。
進入 APP_TaskSwHook 後開始執行 task，
當 Tick = 0 時，TaskID = 1 的優先權為 0，為最高優先權，因此先執行，而 ExecutionTime = 0，執行完透過 OSTimeDly()進入 Wainting state，根據參數設定，delay 3 系統時間。此時發生 task level context switch。如下圖所示:

```
Enter to OS_SchedNew

OS_SchedNew has done
Enter to OSStartHighRdy

enter APP_TaskSwHook

----------------------------------------
Tick: 0, The TaskID = 1
The ArriveTime 0
The ExecutionTime is 0
The TaskPeriodic is 3
The TaskNumber is 0
The priority is 0
----------------------------------------
```

當發生 Task level context switch，會呼叫位在 os_cpu_c.c 中的 OSCtxSw()函式，此函式會將目前 task 的 register 儲存，並載入下一個要做的 task 的 TCB，

```
OSTCBCur = OSTCBHighRdy;
OSPrioCur = OSPrioHighRdy;
```

為了確認流程，將 context switch 發生前後的 task 參數紀錄，如下:

```
The OSTCBCur of current task  = 16902584
The OSPrioCur of current  = 0
The TaskID of current task = 1

The OSTCBCur of task to  run= 16902672
The OSPrioCur of task to run = 1
The TaskID of current task = 2
```

上圖中，上面的三個參數為上一個 task，下面個參數為準備要切換的 task，因此上面執行完後，task1 進入 waiting state，task2 進入 running state，因此如下圖所示:

```
case State created go to task

----------------------------------------
Tick: 0, The TaskID = 2
The ArriveTime 0
The ExecutionTime is 0
The TaskPeriodic is 5
The TaskNumber is 0
The priority is 1
----------------------------------------
```

在 task2 的最後，呼叫 OSTimeDly()，進入 waiting state 並持續 5 系統時間。

此時發生 task level context switch，由於 task1 仍然在 waiting state，因此將由 idle task(declared in os_core.c) 進入 running state，如下圖，可以發現下一個為 idle task

```
Enter to OS_SchedNew

enter task level context switch

enter APP_TaskSwHook


The OSTCBCur of current task  = 10545136
The OSPrioCur of current  = 1
The TaskID of current task = 2

The OSTCBCur of task to  run= 10544960
The OSPrioCur of task to run = 63
The TaskID of current task = 65535
```

在此期間，OS_SchedNew 會一直確認是否有更高優先權的 task 進入 ready state。當 Tick = 3 時，task1 進入 ready state，此時 idle task 仍在 forgearound 執行，因此發生 interrupt level context switch。如下所示:

```
The current task is Idle task

Enter to OS_SchedNew

Enter to OS_SchedNew

Enter to OS_SchedNew

enter Interrput level context switch

enter APP_TaskSwHook


The OSTCBCur of current task  = 10544960
The OSPrioCur of current task = 63
The TaskID of current task = 65535


The OSTCBCur of task to run  = 10545048
The OSPrioCur of task to run = 0
The TaskID of task to run = 1
```

因此將執行 task1，接著 task1 最後執行 OSTimeDly()，進入 waiting state 持續 3 系統時間，而此時 task2 也在 waiting state 中，因此由 idle task 進入 running state。

```
Enter to OS_SchedNew

enter task level context switch

enter APP_TaskSwHook


The OSTCBCur of current task  = 10545048
The OSPrioCur of current  = 0
The TaskID of current task = 1

The OSTCBCur of task to  run= 10544960
The OSPrioCur of task to run = 63
The TaskID of current task = 65535
```

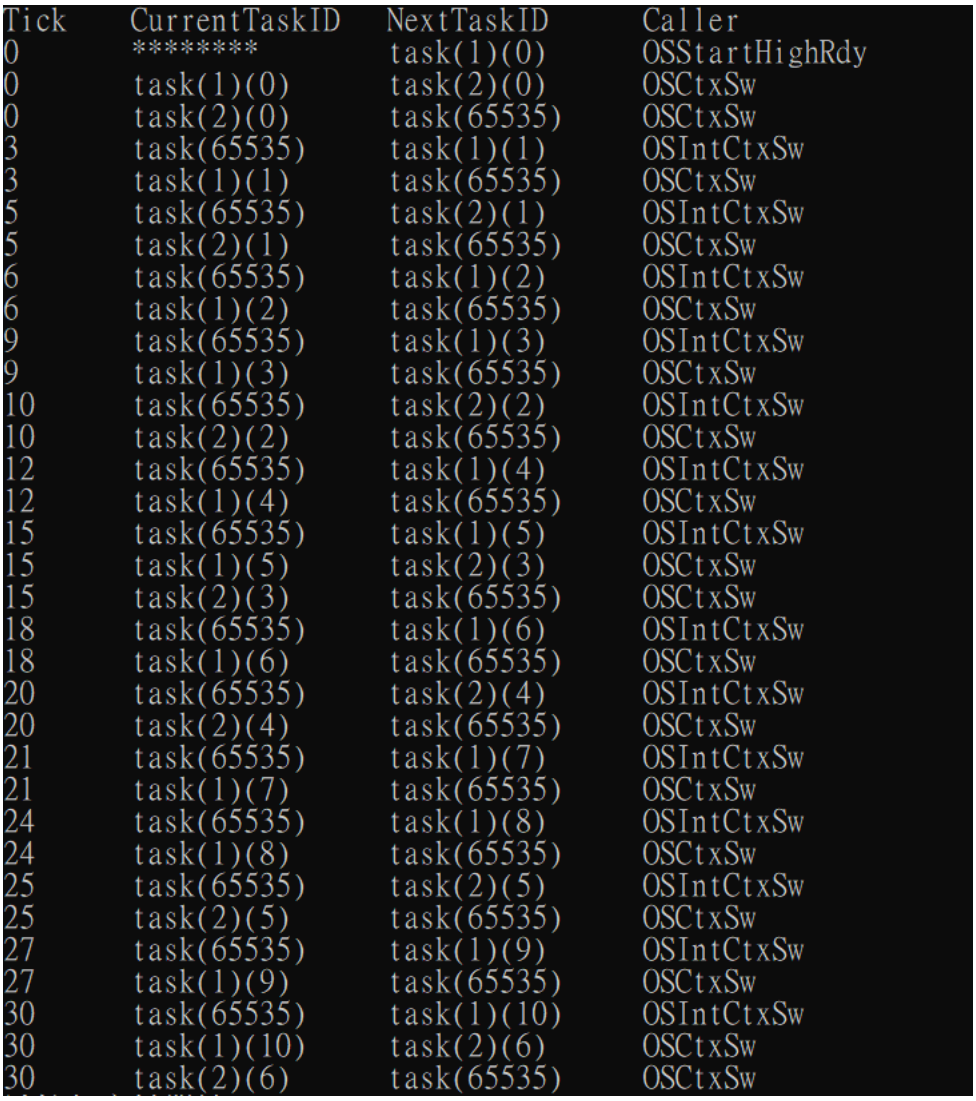
而 OSSchedNew 會一直尋找最高優先權的 task，當 Tick = 5 時，task2 進入 ready state，此時 idle task 仍在 forgearound 執行，因此發生 interrupt level context switch，切換至 task2 執行，如下:

```
Enter to OS_SchedNew

enter Interrput level context switch

enter APP_TaskSwHook


The OSTCBCur of current task  = 10544960
The OSPrioCur of current task = 63
The TaskID of current task = 65535


The OSTCBCur of task to run  = 10545136
The OSPrioCur of task to run = 1
The TaskID of task to run = 2

----------------------------------------
Tick: 5, The TaskID = 2
The ArriveTime 0
The ExecutionTime is 0
The TaskPeriodic is 5
The TaskNumber is 0
The priority is 1
----------------------------------------
```

最後 task2 進入 waiting state，開始重複循環。

(b)

Screenshot Result



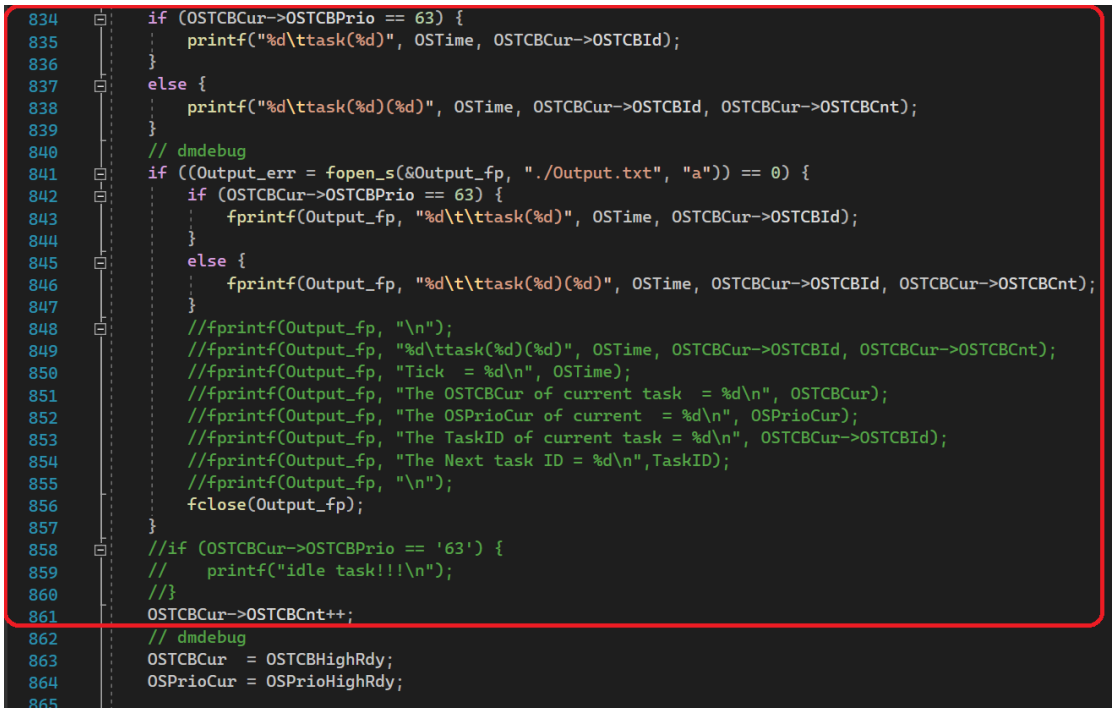
Tick	CurrentTaskID	NextTaskID	Caller
0	********	task(1)(0)	OSStartHighRdy
0	task(1)(0)	task(2)(0)	OSCtxSw
0	task(2)(0)	task(65535)	OSCtxSw
3	task(65535)	task(1)(1)	OSIntCtxSw
3	task(1)(1)	task(65535)	OSCtxSw
5	task(65535)	task(2)(1)	OSIntCtxSw
5	task(2)(1)	task(65535)	OSCtxSw
6	task(65535)	task(1)(2)	OSIntCtxSw
6	task(1)(2)	task(65535)	OSCtxSw
9	task(65535)	task(1)(3)	OSIntCtxSw
9	task(1)(3)	task(65535)	OSCtxSw
10	task(65535)	task(2)(2)	OSIntCtxSw
10	task(2)(2)	task(65535)	OSCtxSw
12	task(65535)	task(1)(4)	OSIntCtxSw
12	task(1)(4)	task(65535)	OSCtxSw
15	task(65535)	task(1)(5)	OSIntCtxSw
15	task(1)(5)	task(2)(3)	OSCtxSw
15	task(2)(3)	task(65535)	OSCtxSw
18	task(65535)	task(1)(6)	OSIntCtxSw
18	task(1)(6)	task(65535)	OSCtxSw
20	task(65535)	task(2)(4)	OSIntCtxSw
20	task(2)(4)	task(65535)	OSCtxSw
21	task(65535)	task(1)(7)	OSIntCtxSw
21	task(1)(7)	task(65535)	OSCtxSw
24	task(65535)	task(1)(8)	OSIntCtxSw
24	task(1)(8)	task(65535)	OSCtxSw
25	task(65535)	task(2)(5)	OSIntCtxSw
25	task(2)(5)	task(65535)	OSCtxSw
27	task(65535)	task(1)(9)	OSIntCtxSw
27	task(1)(9)	task(65535)	OSCtxSw
30	task(65535)	task(1)(10)	OSIntCtxSw
30	task(1)(10)	task(2)(6)	OSCtxSw
30	task(2)(6)	task(65535)	OSCtxSw

Tick	CurrentTaskID	NextTaskID	Caller
0	********	task(1)(0)	OSStartHighRdy
0	task(1)(0)	task(2)(0)	OSCtxSw
0	task(2)(0)	task(65535)	OSCtxSw
3	task(65535)	task(1)(1)	OSIntCtxSw
3	task(1)(1)	task(65535)	OSCtxSw
5	task(65535)	task(2)(1)	OSIntCtxSw
5	task(2)(1)	task(65535)	OSCtxSw
6	task(65535)	task(1)(2)	OSIntCtxSw
6	task(1)(2)	task(65535)	OSCtxSw
9	task(65535)	task(1)(3)	OSIntCtxSw
9	task(1)(3)	task(65535)	OSCtxSw
10	task(65535)	task(2)(2)	OSIntCtxSw
10	task(2)(2)	task(65535)	OSCtxSw
12	task(65535)	task(1)(4)	OSIntCtxSw
12	task(1)(4)	task(65535)	OSCtxSw
15	task(65535)	task(1)(5)	OSIntCtxSw
15	task(1)(5)	task(2)(3)	OSCtxSw
15	task(2)(3)	task(65535)	OSCtxSw
18	task(65535)	task(1)(6)	OSIntCtxSw
18	task(1)(6)	task(65535)	OSCtxSw
20	task(65535)	task(2)(4)	OSIntCtxSw
20	task(2)(4)	task(65535)	OSCtxSw
21	task(65535)	task(1)(7)	OSIntCtxSw
21	task(1)(7)	task(65535)	OSCtxSw
24	task(65535)	task(1)(8)	OSIntCtxSw
24	task(1)(8)	task(65535)	OSCtxSw
25	task(65535)	task(2)(5)	OSIntCtxSw
25	task(2)(5)	task(65535)	OSCtxSw
27	task(65535)	task(1)(9)	OSIntCtxSw
27	task(1)(9)	task(65535)	OSCtxSw
30	task(65535)	task(1)(10)	OSIntCtxSw
30	task(1)(10)	task(2)(6)	OSCtxSw
30	task(2)(6)	task(65535)	OSCtxSw

Explanation

為了將 current taskID 和 next taskID 紀錄並印出來，必須要找到這兩者之間交換的地方，也就是 context switch 發生的地方。而在 uc/OS 2 中 context switch 有分成 task level 和 interrupted level 兩種，因此更改這兩個區塊的程式碼即可。

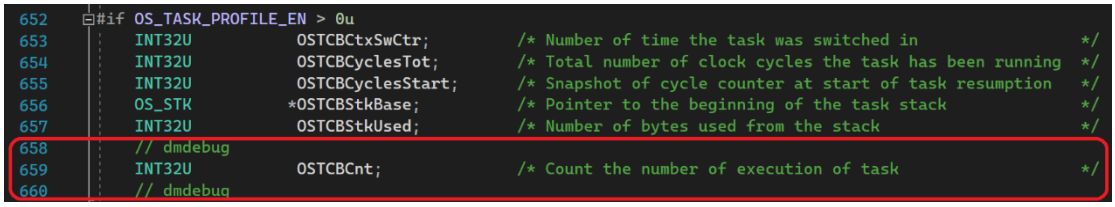
操作 Task level context switch 的函式為 OSCtxSw，定義在 os_cpu_c.c 檔案中，為了找出 current taskID，必須要在 OSTCBCur 的指標和 OSPrioCur 改變前，新增 code 如下:



```
834     if (OSTCBCur->OSTCBPrio == 63) {
835         printf("%d\ttask(%d)", OSTime, OSTCBCur->OSTCBId);
836     }
837     else {
838         printf("%d\ttask(%d)(%d)", OSTime, OSTCBCur->OSTCBId, OSTCBCur->OSTCBCnt);
839     }
840     // dmdebug
841     if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
842         if (OSTCBCur->OSTCBPrio == 63) {
843             fprintf(Output_fp, "%d\t\ttask(%d)", OSTime, OSTCBCur->OSTCBId);
844         }
845         else {
846             fprintf(Output_fp, "%d\t\ttask(%d)(%d)", OSTime, OSTCBCur->OSTCBId, OSTCBCur->OSTCBCnt);
847         }
848         //fprintf(Output_fp, "\n");
849         //fprintf(Output_fp, "%d\ttask(%d)(%d)", OSTime, OSTCBCur->OSTCBId, OSTCBCur->OSTCBCnt);
850         //fprintf(Output_fp, "Tick  = %d\n", OSTime);
851         //fprintf(Output_fp, "The OSTCBCur of current task  = %d\n", OSTCBCur);
852         //fprintf(Output_fp, "The OSPrioCur of current  = %d\n", OSPrioCur);
853         //fprintf(Output_fp, "The TaskID of current task = %d\n", OSTCBCur->OSTCBId);
854         //fprintf(Output_fp, "The Next task ID = %d\n",TaskID);
855         //fprintf(Output_fp, "\n");
856         fclose(Output_fp);
857     }
858     //if (OSTCBCur->OSTCBPrio == '63') {
859     //    printf("idle task!!!\n");
860     //}
861     OSTCBCur->OSTCBCnt++;
862     // dmdebug
863     OSTCBCur  = OSTCBHighRdy;
864     OSPrioCur = OSPrioHighRdy;
```

第 834 行 ～ 第 839 行：判斷目前的 task 是否為 idle task，以 idle taskID 為 63 當作判斷條件。
第 841 行 ～ 第 847 行：與上述相同，差別在於這次是要寫入檔案。
第 861 行：將 OSTCBCnt 加 1，用來記錄這個 task 的執行次數。(下面說明)

為了記錄每個 Task 的執行次數，在 TCB 新增變數 OSTCBCnt，TCB 資料結構定義在 ucos_ii.h 中，如下:



```
652 #if OS_TASK_PROFILE_EN > 0u
653     INT32U          OSTCBCtxSwCtr;          /* Number of time the task was switched in                */
654     INT32U          OSTCBCyclesTot;         /* Total number of clock cycles the task has been running */
655     INT32U          OSTCBCyclesStart;       /* Snapshot of cycle counter at start of task resumption  */
656     OS_STK         *OSTCBStkBase;           /* Pointer to the beginning of the task stack             */
657     INT32U          OSTCBStkUsed;           /* Number of bytes used from the stack                    */
658     // dmdebug
659     INT32U          OSTCBCnt;               /* Count the number of execution of task                  */
660     // dmdebug
```

(若有自行修改之區域，會在修改區域上下新增註解已表示，如// dmdebug)。

接著 OSTCBCur 和 OSPrioCur 就會更新到下一個要執行的 task，再將上述的程式碼複製到此即可得到 next taskID，如下:

```
863     OSTCBCur  = OSTCBHighRdy;
864     OSPrioCur = OSPrioHighRdy;
865
866     // dmdebug
867     //printf("The OSTCBCur = %d\n", OSTCBCur);
868     //printf("The OSPrioCur = %d\n", OSPrioCur);
869     //printf("cnt = %d\n", OSTCBCur->OSTCBCnt);
870     // dmdebug
871     if (OSTCBCur->OSTCBPrio == 63) {
872         printf("\ttask(%d)\tOSCtxSw\n", OSTCBCur->OSTCBId);
873     }
874     else {
875         printf("\ttask(%d)(%d)\tOSCtxSw\n", OSTCBCur->OSTCBId, OSTCBCur->OSTCBCnt);
876     }
877     if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
878         if (OSTCBCur->OSTCBPrio == 63) {
879             fprintf(Output_fp, "\t\ttask(%d)\t\tOSCtxSw\n", OSTCBCur->OSTCBId);
880         }
881         else {
882             fprintf(Output_fp, "\t\ttask(%d)(%d)\t\tOSCtxSw\n", OSTCBCur->OSTCBId, OSTCBCur->OSTCBCnt);
883         }
884        //fprintf(Output_fp, "\ttask(%d)(%d)\n", OSTCBCur->OSTCBId, OSTCBCur->OSTCBCnt);
885        // fprintf(Output_fp, "Tick  = %d\n", OSTime);
886        // fprintf(Output_fp, "The OSTCBCur of task to  run= %d\n", OSTCBCur);
887        // fprintf(Output_fp, "The OSPrioCur of task to run = %d\n", OSPrioCur);
888        // fprintf(Output_fp, "The TaskID of current task = %d\n", OSTCBCur->OSTCBId);
889         //fprintf(Output_fp, "The Next task ID = %d\n",TaskID);
890        // fprintf(Output_fp, "\n");
891         fclose(Output_fp);
892     }
```

第 871 行～第 891 行：判斷是否為 idle task，並將 next taskID 和 call function 印出，由於是 OSCtxSw 切換至下一個 task，因此印出 OSCtxSw。

Interrupt level context switch 函示 OSIntCtxSw 也定義在 os_cpu_c.c，位子就在 task level context switch(OSCtxSw)下，程式碼看 task level 基本相同。

```
1050    if (OSTCBCur->OSTCBPrio == 63) {
1051        printf("%d\ttask(%d)", OSTime, OSTCBCur->OSTCBId);
1052    }
1053    else {
1054        printf("%d\ttask(%d)(%d)", OSTime, OSTCBCur->OSTCBId, OSTCBCur->OSTCBCnt);
1055    }
1056    //printf("%d\ttask(%d)(%d)", OSTime, OSTCBCur->OSTCBId, OSTCBCur->OSTCBCnt);
1057    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
1058        if (OSTCBCur->OSTCBPrio == 63) {
1059            fprintf(Output_fp, "%d\t\ttask(%d)", OSTime, OSTCBCur->OSTCBId);
1060        }
1061        else {
1062            fprintf(Output_fp, "%d\t\ttask(%d)(%d)", OSTime, OSTCBCur->OSTCBId, OSTCBCur->OSTCBCnt);
1063        }
1064        //fprintf(Output_fp, "%d\ttask(%d)(%d)", OSTime, OSTCBCur->OSTCBId, OSTCBCur->OSTCBCnt);
1065       // fprintf(Output_fp, "\n");
1066       // fprintf(Output_fp, "Tick  = %d\n", OSTime);
1067       // fprintf(Output_fp, "The OSTCBCur of current task  = %d\n", OSTCBCur);
1068       // fprintf(Output_fp, "The OSPrioCur of current task = %d\n", OSPrioCur);
1069       // fprintf(Output_fp, "The TaskID of current task = %d\n", OSTCBCur->OSTCBId);
1070        //fprintf(Output_fp, "The Next task ID = %d\n",TaskID);
1071      //  fprintf(Output_fp, "\n");
1072        fclose(Output_fp);
1073    }
1074    OSTCBCur->OSTCBCnt++;
```

在 OSTCBCur 和 OSTPrioCur 更新後再做一次

```
1076    OSTCBCur  = OSTCBHighRdy;
1077    OSPrioCur = OSPrioHighRdy;
1078
1079    // dmdebug
1080    if (OSTCBCur->OSTCBPrio == 63) {
1081        printf("\ttask(%d)\tOSIntCtxSw\n", OSTCBCur->OSTCBId);
1082    }
1083    else {
1084        printf("\ttask(%d)(%d)\tOSIntCtxSw\n", OSTCBCur->OSTCBId, OSTCBCur->OSTCBCnt);
1085    }
1086    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
1087        if (OSTCBCur->OSTCBPrio == 63) {
1088            fprintf(Output_fp, "\t\ttask(%d)\t\tOSIntCtxSw\n", OSTCBCur->OSTCBId);
1089        }
1090        else {
1091            fprintf(Output_fp, "\t\ttask(%d)(%d)\t\tOSIntCtxSw\n", OSTCBCur->OSTCBId, OSTCBCur->OSTCBCnt);
1092        }
1093        //fprintf(Output_fp, "\ttask(%d)(%d)\n", OSTCBCur->OSTCBId, OSTCBCur->OSTCBCnt);
1094        //fprintf(Output_fp, "\n");
1095        //fprintf(Output_fp, "Tick  = %d\n", OSTime);
1096        //fprintf(Output_fp, "The OSTCBCur of task to run  = %d\n", OSTCBCur);
1097        //fprintf(Output_fp, "The OSPrioCur of task to run = %d\n", OSPrioCur);
1098        //fprintf(Output_fp, "The TaskID of task to run = %d\n", OSTCBCur->OSTCBId);
1099        //fprintf(Output_fp, "The Next task ID = %d\n",TaskID);
1100        //fprintf(Output_fp, "\n");
1101        fclose(Output_fp);
1102    }
```

結合上述，就能得到所有的 Current taskID 和 next taskID。但是當 Tick = 0 時，task level 和 interrupt level 都沒有被觸發。要找出 next taskID 就無法透過上述方法。

當 Tick = 0 時，OSS_chedNew()會找出最高優先權的 task，並由 OSStartHighRdy() 會執行，因此可以在 OSStartHighRdy() 找到一開始所要執行的 next taskID，如下:

```
682 void  OSStartHighRdy (void)
683 {
684     OS_TASK_STK  *p_stk;
685     OS_TCB       *p_tcb;
686     INT8U         prio;
687     CPU_SR_ALLOC();
688
689     // dmdebug
690     //printf("Enter to OSStartHighRdy\n");
691     //printf("\n");
692    // if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
693         fprintf(Output_fp, "Enter to OSStartHighRdy\n");
694         fprintf(Output_fp, "\n");
695         fclose(Output_fp);
696    // }
697     // dmdebug
698
699     OSTaskSwHook(); // (a)
700     OSRunning = 1;  // (b)
701
702     p_stk = (OS_TASK_STK *)OSTCBHighRdy->OSTCBStkPtr;                    /* OSTCBCur  = OSTCBHighRdy;
703                                                                          /* OSPrioCur = OSPrioHighRdy;
704     // the above is (c)
705     // dmdebug
706     printf("\ttask(%d)(%d)\tOSStartHighRdy\n",OSTCBCur->OSTCBId, OSTCBCur->OSTCBCnt);
707     if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
708         fprintf(Output_fp, "\t\ttask(%d)(%d)\t\tOSStartHighRdy\n", OSTCBCur->OSTCBId, OSTCBCur->OSTCBCnt);
709         fclose(Output_fp);
710     }
```

第 706 行～第 710 行：印出下一個 taskID，由於 OSStartHighRdy 只會執行一次，因此之後並不會再出現同樣的 call function。