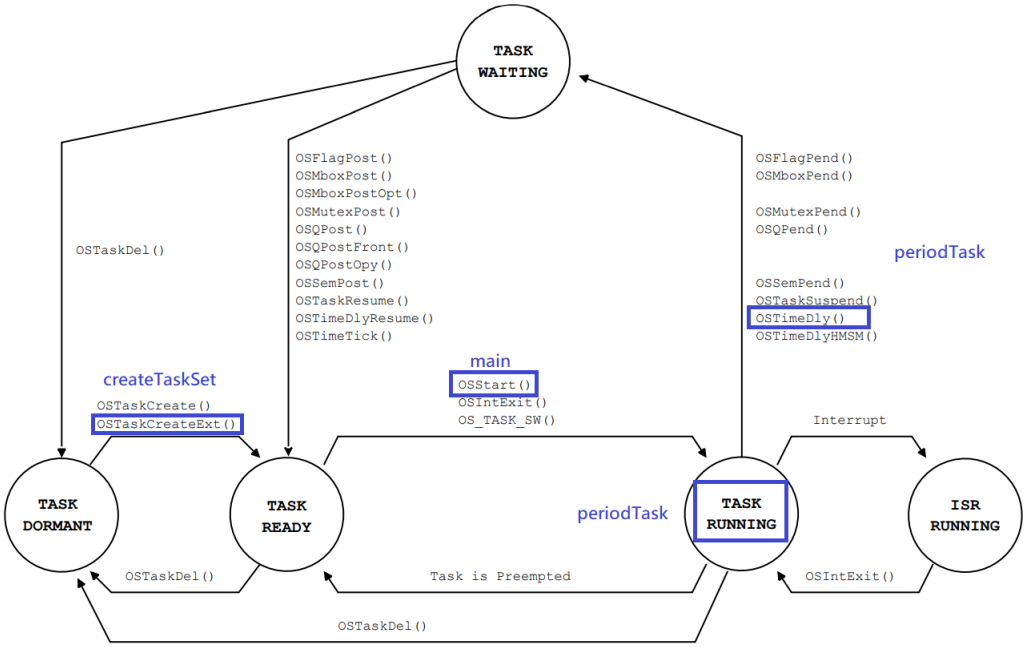
# 作業系統實務期末報告 - group 5

組員 :

| 姓名學號 | 分工比例 |
| --- | --- |
| 蘇青衛B0629010 | 25% |
| 顏于婷B0629023 | 25% |
| 江行之B0629038 | 25% |
| 謝秉寰B0629041 | 25% |

1. RM
    1. 每個函數之間的關係
        1. main.c檔
            1. main
               負責初始化作業系統相關資源後，呼叫`createTaskSet`建立任務，然後再呼叫`OSStart`開始執行多任務
            2. createTaskSet
               負責讀檔得到任務數量與執行時間和週期等資訊後建立週期性任務，會根據依序挑出週期最小的工作，以遞增優先權值的方式建立任務(呼a叫`OSTaskCreateExt`)
            3. periodTask

               一個任務會執行的函數，內部就是無窮迴圈，當任務執行的剩餘時間歸零時，會重新計算deadline、將剩餘時間回復到滿，然後呼叫`OSTimeDly`變成waiting狀態的任務直到時間到才變回ready狀態
        2. os_core.c檔
            1. OSTimeTick
               每次每一執行1 tick的時間就會呼叫此函數，需要把目前執行的週期性任務的`CompTime`(執行剩餘時間)減1，以及遍歷所有任務將有`OSTCBDly`的任務的`OSTCBDly--`(`OSTimeDly`延遲的剩餘時間)，並且將`OSTCBDly`為0的工作變為ready狀態
            2. OS_Sched
               會判斷目前最高優先權的工作，如果最高優先權的工作不是當前執行的工作就做context switch
            3. OS_TCBInit
               每個任務都會有它的TCB(task control block)，紀錄一些任務的相關資訊，像是要delay多久才加到ready list、目前的狀態、優先度、任務的堆疊指標、任務的id等資訊。
               這邊會新增幾個新的欄位，有`CompTime`(剩餘執行時間)、`StartTime`開始執行時間)、`Deadline`(截止時間)，如果是extend task就會有這些欄位
    2. 實作的流程圖




上圖是函數跟任務狀態切換的關係

一開始的main函數會做作業系統的初始化，然後呼叫`createTaskSet`建立週期性任務(`OSTaskCreateExt`)，如圖所示，`createTaskSet`會讓新建的任務在ready狀態，以及根據週期指定優先度，後面在呼叫`OSStart`挑選一個最高優先權的任務變成task running狀態，執行的任務會執行`periodTask`，`periodTask`的執行時間做完後會重新計算deadline等資訊以及要休息的時間然後呼叫`OSTimeDly`休息到deadline再變成task ready狀態跟著其他任務競爭cpu

3. 實作的細節(講解部分在註解)

   修改OS_TCB的結構，加入`CompTime`(剩餘執行時間)、`StartTime`開始執行時間)、`Deadline`(截止時間)

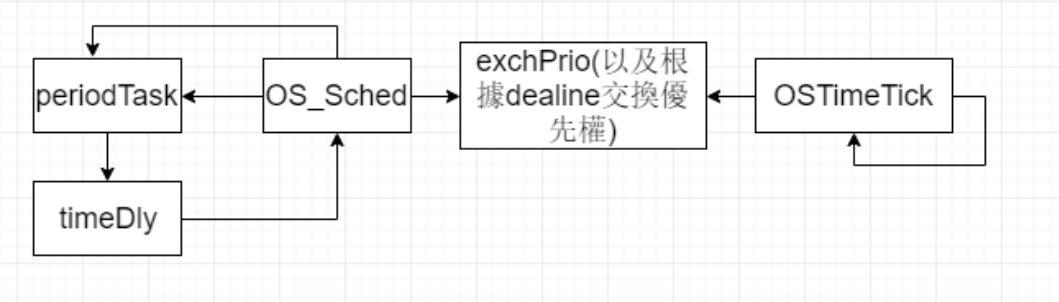
```
typedef struct os_tcb {
// 前略
    INT32U           CompTime;              /* 剩餘執行時間 */
    INT32U           StartTime;             /* 開始時間 */
    INT32U           Deadline;              /* 截止時間 */
// 後略
}
```

   依序挑選最短周期的欲建立任務，然後以高到低優先權依序建立

```
for (int i = 0; i < NumberOfTasks; i++)
{
    int minPeriod = 100000000;
    int minPeriodTaskIndex = 0;
    for (int j = 0; j < NumberOfTasks; j++) // 遍歷找到還沒有被挑的最小週期任務
    {
        if (taskTimeInfos[i][2]-taskTimeInfos[i][1] < minPeriod) {
            int hasIn = 0;
            for (int k = 0; k <= i; k++) // 檢查此任務是否被挑過
            {
                if (hasAssignedTask[k] == j) {
                    hasIn = 1;
                }
            }
```

```
            if (hasIn == 0) { // 沒有被挑過就紀錄目前的暫時是最小週期的任務
                minPeriod = taskTimeInfos[j][2] - taskTimeInfos[i][1];
                minPeriodTaskIndex = j;
                hasAssignedTask[i] = j;
            }
        }
    }
    OSTaskCreateExt(...); // 建立被挑的那個任務，優先權從20遞增(數字越小越優先)
}
```

2. EDF(有一些部份跟RM相同就不多贅述)
    1. 每個函數之間的關係
        1. main.c檔(跟RM相似，只差在挑deadline最小的而不是挑period最小的)
        2. os_core.c檔
            1. OSTimeTick
                1. 將週期性任務的CompTime-1
                2. 要根據deadline交換優先權(呼叫`OSTaskChangePrio`)
            2. OS_Sched
                1. 要根據deadline交換優先權再排程(呼叫`OSTaskChangePrio`)
                2. 觸發時機: 1. 當週期性任務`periodTask`呼叫`timeDly`時，會將其移出readylist並且呼叫`OS_Sched` 2. 當週期性任務`periodTask`呼叫`timeDly`之後時間到，會在`OSTimeTick`觸發`OS_Sched`
            3. OSTaskChangePrio(自己定義的函數)
               將兩個任務的優先權交換，在OSTImeTick和OS_Sched時要呼叫，因為要挑最早的deadline任務來做
        3. os_task.c檔
            1. OSTaskChangePrio
               將原本某個優先權的任務換成另外一個優先權，會改到優先權與OS_TCB的表~`OSTCBPrioTbl`以及準備執行任務的表~`OSRdyTbl`。然後`OSTaskChangePrio`會呼叫此函數來完成兩任務優先權交換。將呼叫OS_Sched的部分註解掉。
    2. 實作的流程圖




    3. 實作的細節(講解部分在註解)

       OSTimeTick和OS_Sched的根據deadline交換優先權的部分

```
OS_TCB* ptcb = OSTCBList; // 得到指向OS_TCB list的第一個任務的tcb
OS_TCB* hptcb = OSTCBPrioTbl[PERIODIC_TASK_START_PRIO]; // 得到指向目前
priority最高的TCB指標
while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {  // 把所有任務走過一遍
    // 確定此任務是週期性工作且不在waiting
    if (ptcb->OSTCBDly == 0u && ptcb->OSTCBExtPtr != 0 && hptcb-
>OSTCBExtPtr != 0) {
```

```
        // 只要最高優先權的工作有delay且不是目前遍歷的任務就交換優先權，把最高優先
權的工作換掉
        if (hptcb->OSTCBDly != 0u && hptcb != ptcb) {

            exchPrio(hptcb->OSTCBPrio, ptcb->OSTCBPrio);
            hptcb = ptcb;
        }
        // 如果兩者deadline相同就看其他條件決定要不要交換
        if (ptcb->Deadline == hptcb->Deadline) {
            INT32U restTime = ptcb->Deadline - OSTimeGet();
            // 如果在deadline前兩者都能做完，就ID小的先做，如果目前遍歷的任務ID較
小就交換優先權
            if ((ptcb->CompTime + hptcb->CompTime) > restTime) {

                if (ptcb->OSTCBId < hptcb->OSTCBId) {
                    exchPrio(hptcb->OSTCBPrio, ptcb->OSTCBPrio);
                    hptcb = ptcb;
                }

            } // 如果deadline前只有一個能做完就挑最早能執行的
            else if (ptcb->StartTime < hptcb->StartTime) {
                exchPrio(hptcb->OSTCBPrio, ptcb->OSTCBPrio);
                hptcb = ptcb;
            }
        } // 如果目前遍歷的任務的deadline比高優先權的早就兩者對換
        else if (ptcb->Deadline < hptcb->Deadline) {

            exchPrio(hptcb->OSTCBPrio, ptcb->OSTCBPrio);
            hptcb = ptcb;

        }
    }
    ptcb = ptcb->OSTCBNext;                         // 將目前遍歷的任務改成下
一個
}
```

exchPrio

```
INT8U tempPrio = 59u;

OSTaskChangePrio(prio1, tempPrio); // 先把prio1的任務換到tempPrio的位置
OSTaskChangePrio(prio2, prio1);// 先把prio2的任務換到prio1的位置
OSTaskChangePrio(tempPrio, prio2); // 最後把tempPrio的任務換到prio2的位置
```