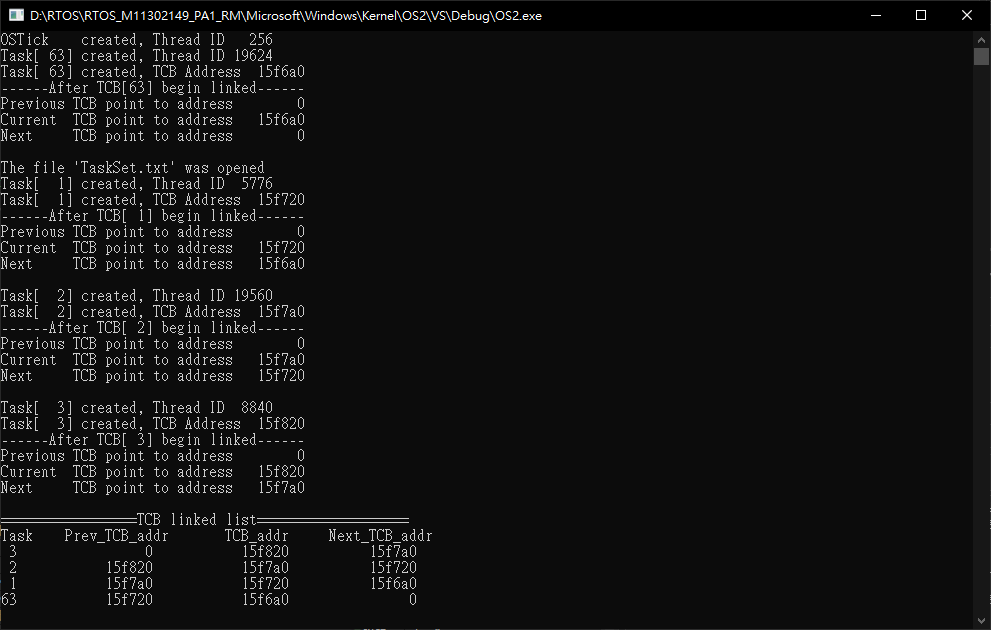
RTOS PA #1 Report

Name: 趙孟哲  
Student ID：M11302149

[ PART I ] Task Control Block Linked List

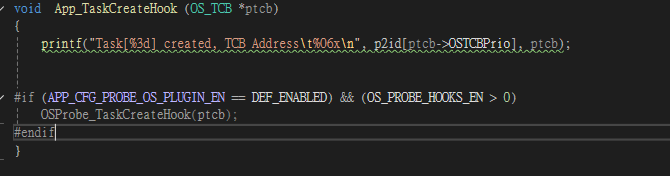
1. The screenshot results. (10%)



1. A report that describes your implementation(10%).

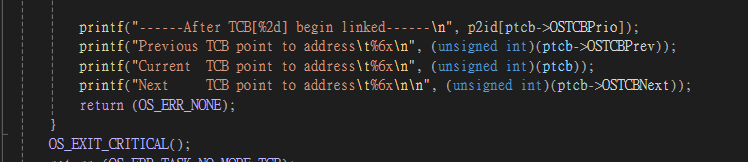
本部分主要在 μC/OS-II 核心中加入額外輸出，用以觀察 TCB（Task Control Block）以及其linked list之狀態。  
我所修改的程式位於，app\_hook.c的App\_taskHook()中當task創建時印出TCB位址。





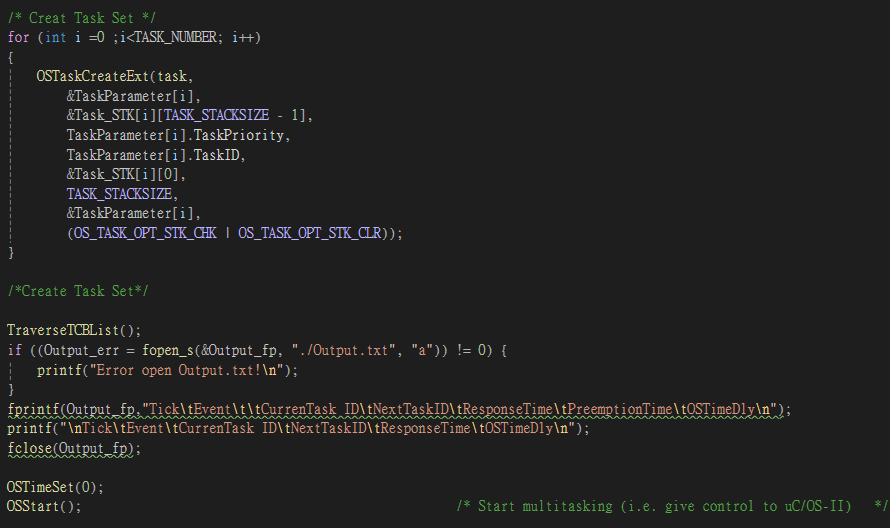


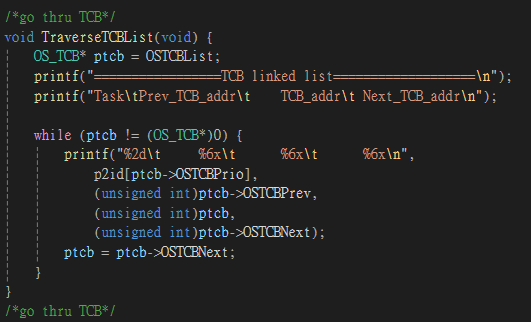
在os\_core.c 的OSTCBInit()最後，於插入 TCB 至OSTCBList 時，額外輸出目前TCB 的位址，以及前後指向的位址



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。







可以知道：OSTCB指向最後一個被創建的TCB，idle 最先被創建。

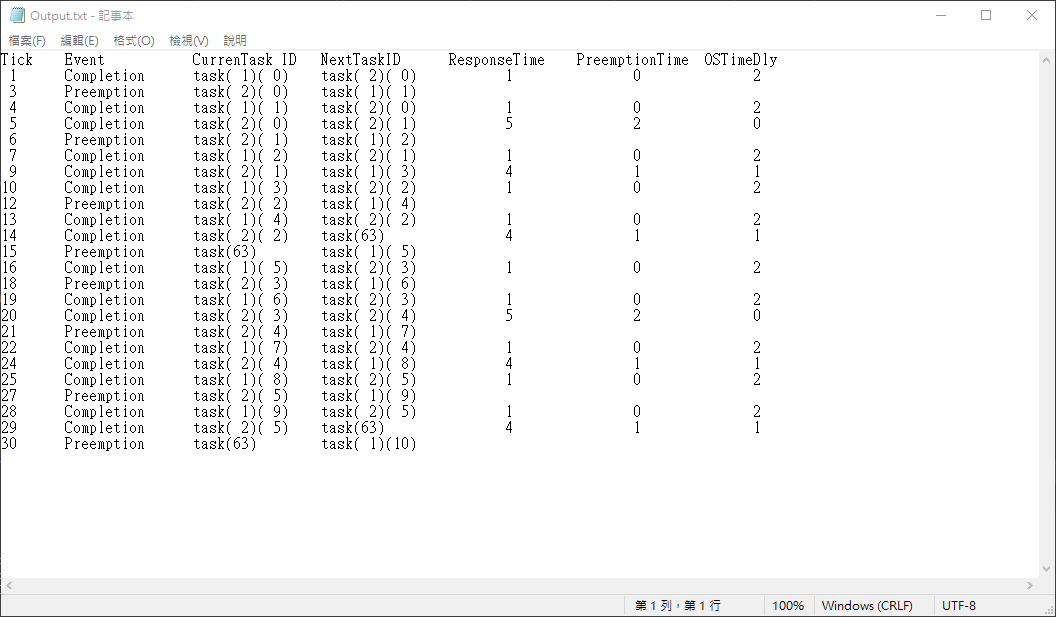
[ PART II ] RM Scheduler Implementation [70%]

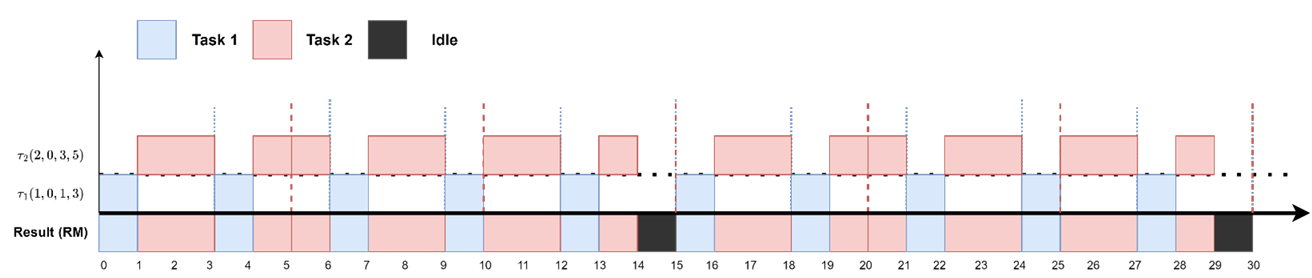
1. Schedule results of examples. (25%)

Taskset I

1 0 1 3

2 0 3 5



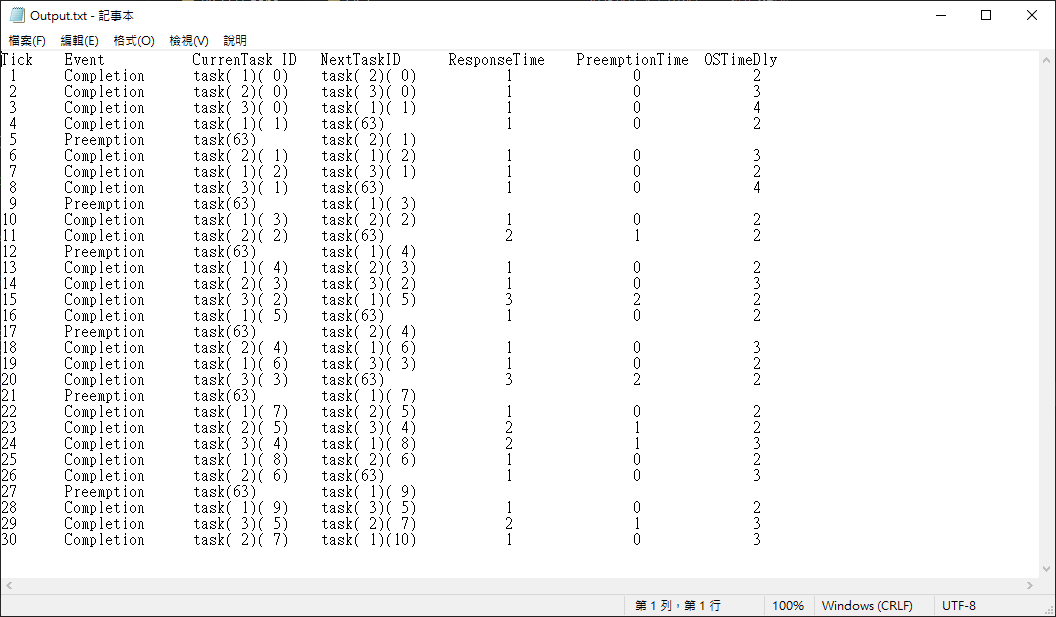


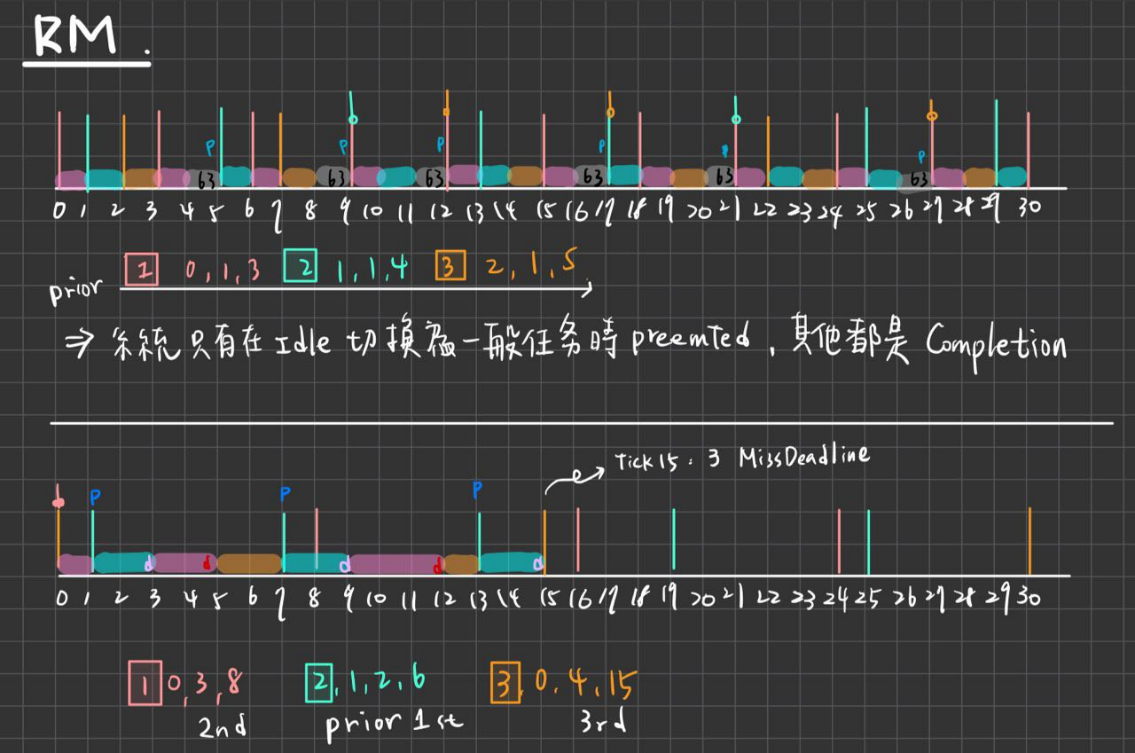
Taskset II

1 0 1 3

2 1 1 4

3 2 1 5



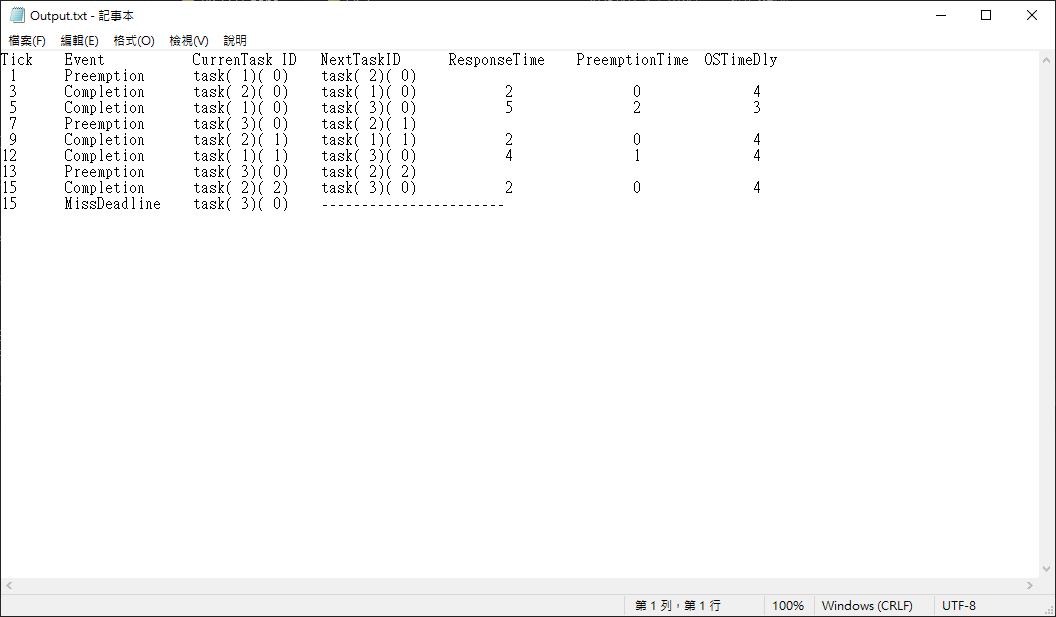


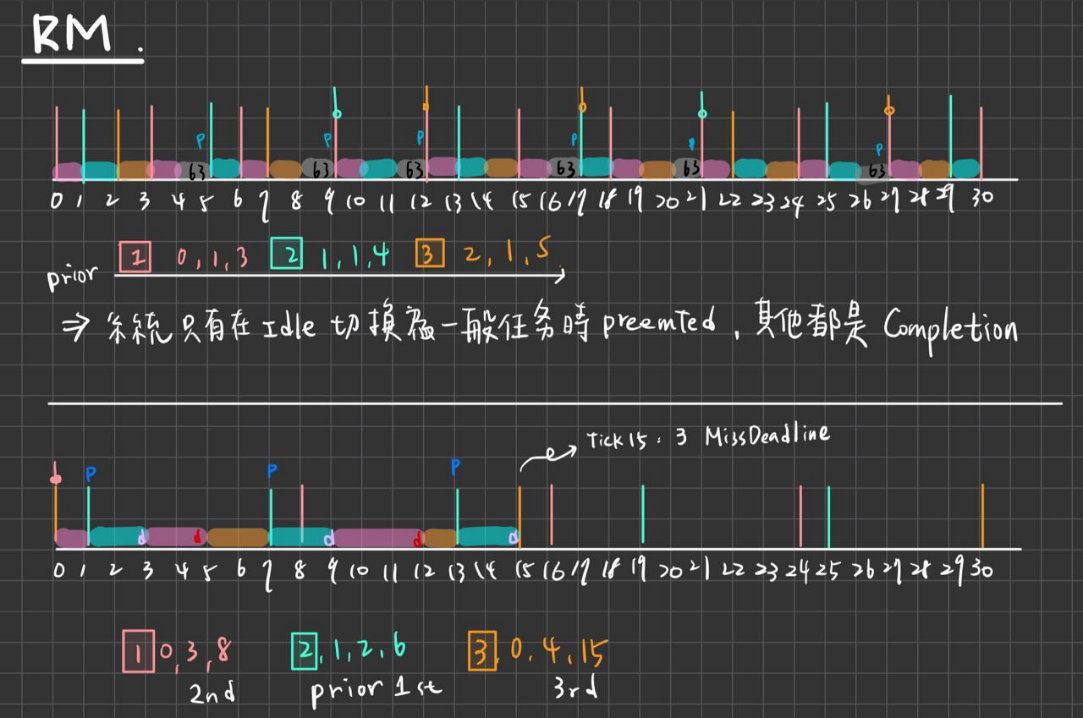
Taskset III

1 0 3 8

2 1 2 6

3 0 4 15

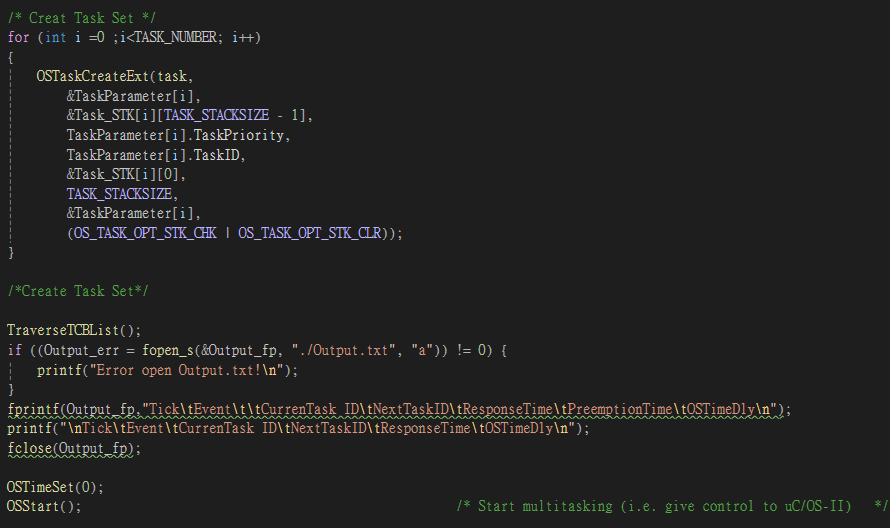




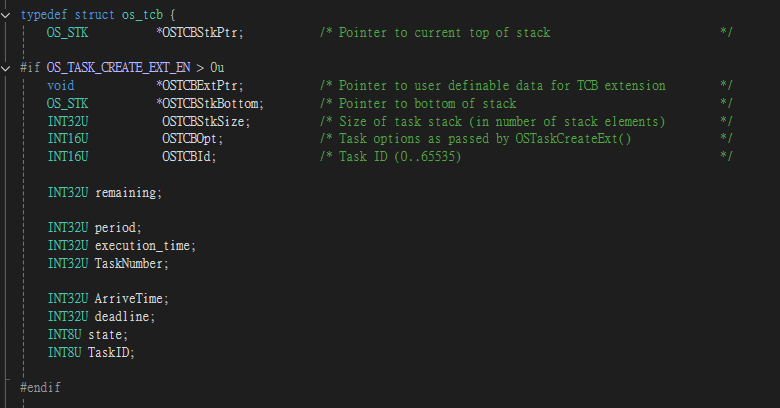
1. Implementation(40%)

首先印出標題列



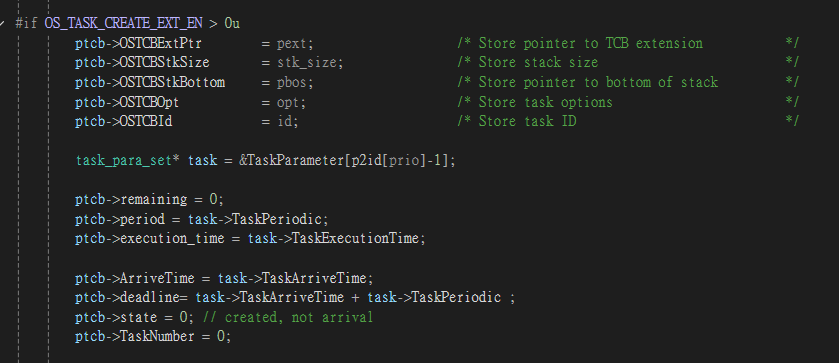


在ucosii.h 的OS\_TCB中，每個TCB新增task執行狀況的欄位。



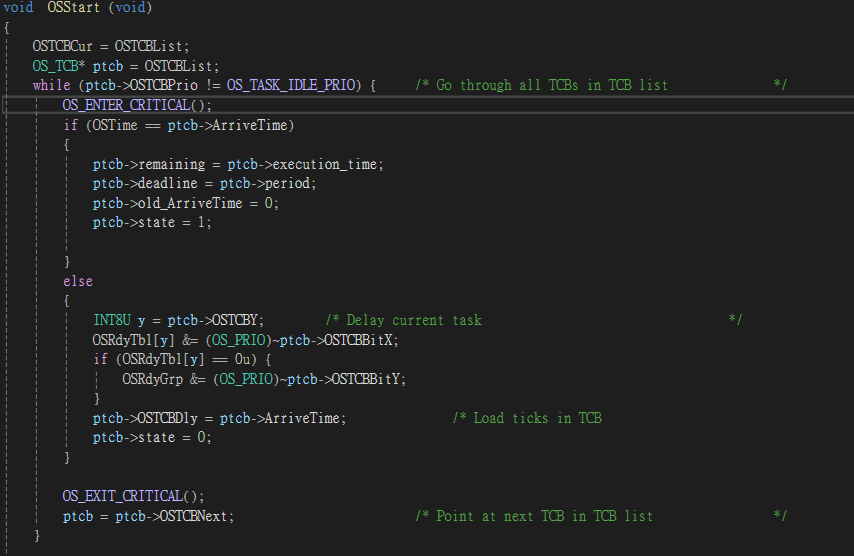
OS\_TCBInit 初始化TCB



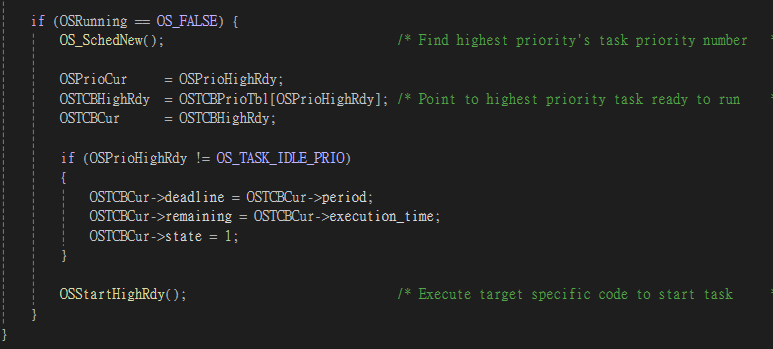


OsStart() 要設定tick=0時的狀態：除了arrival 的task之外其他設置OSTCBDly，將task 從ready table中移除(因為是設task創建時會設為ready。

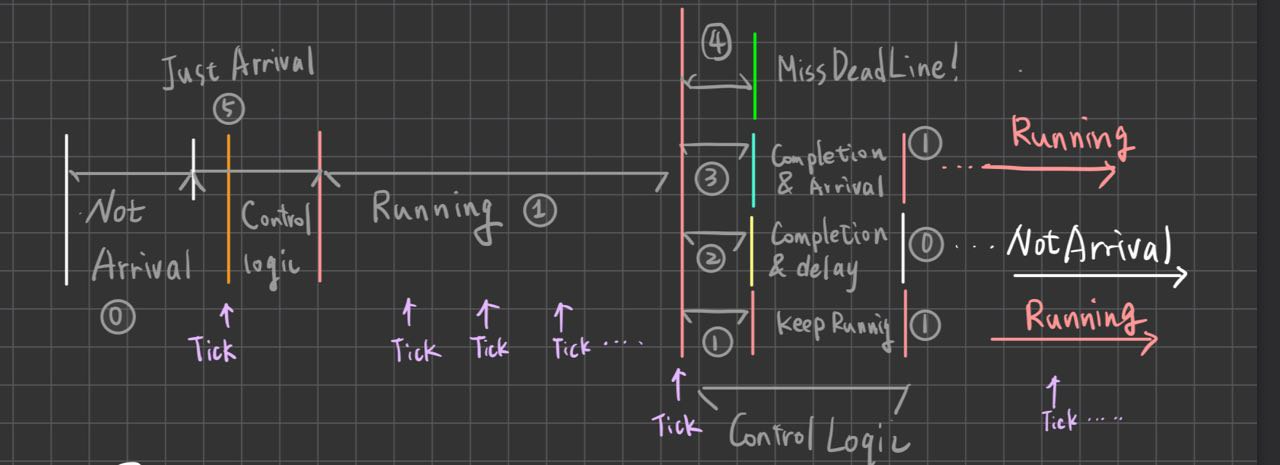
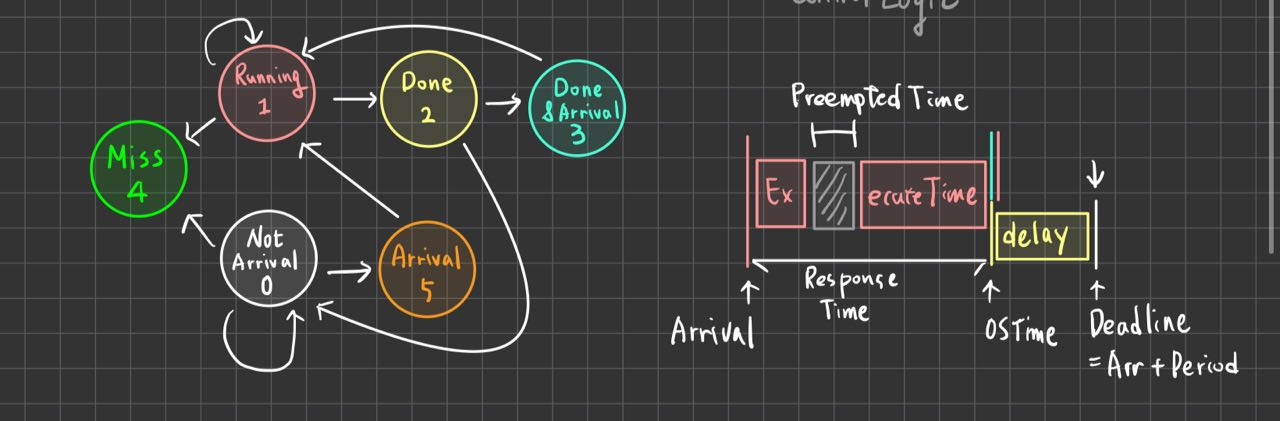




呼叫sched\_new()選出highRdy的task，設定執行的參數，例如執行時間、任務狀態和DeadLine ，然後呼叫OSStartHighRdy開始執行highRdy task。

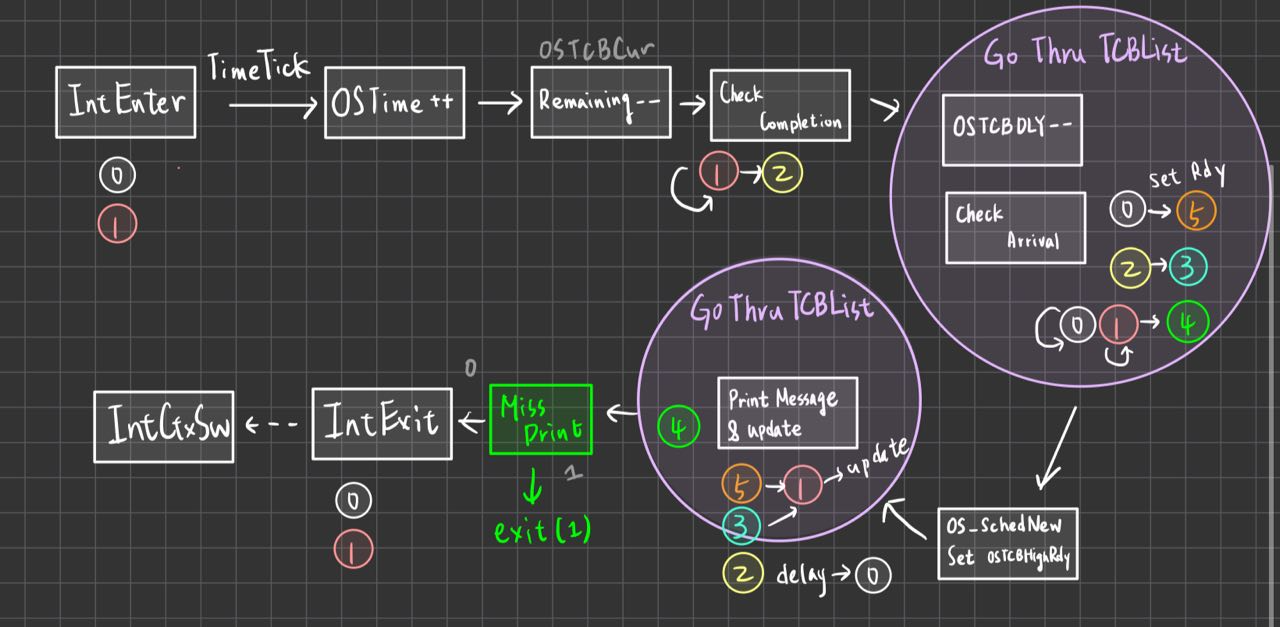


我使用FSM監控task的執行狀態。



* 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。

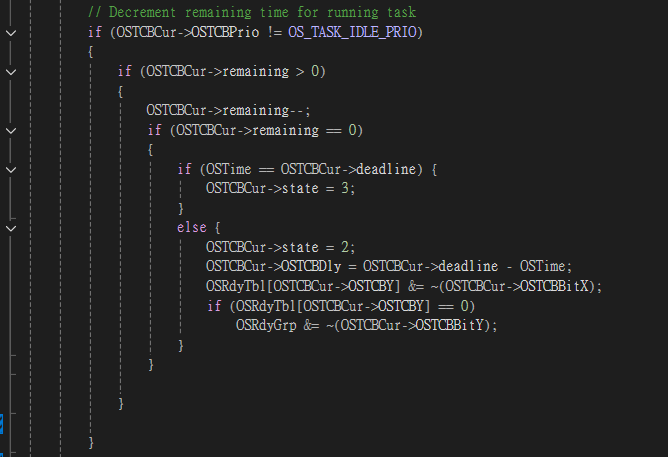
以下是OStimetick的運作流程圖。



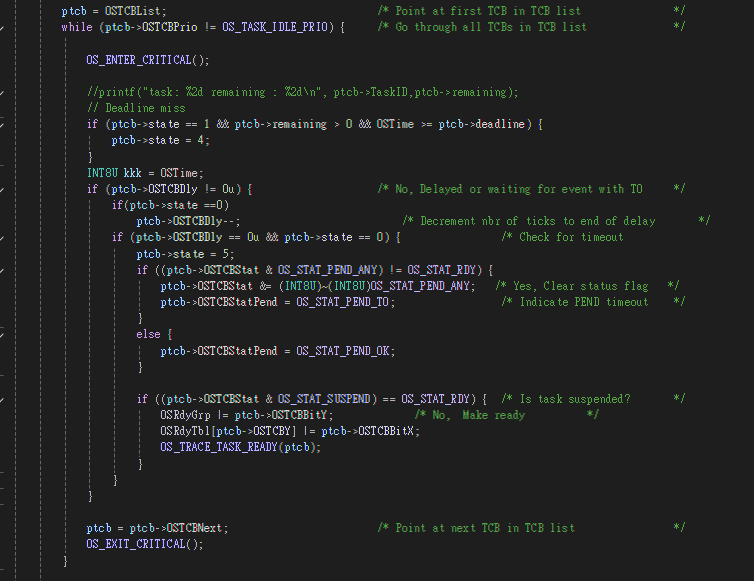
依照 RM 排程規則，以任務Period為依據決定優先權，週期越短優先權越高。我在OSTimeTick()實作task 運作邏輯。首先將目前執行的任務(idle忽略)的remaining遞減。如果變為0，檢查是否到下次週期，區分狀態。

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）。





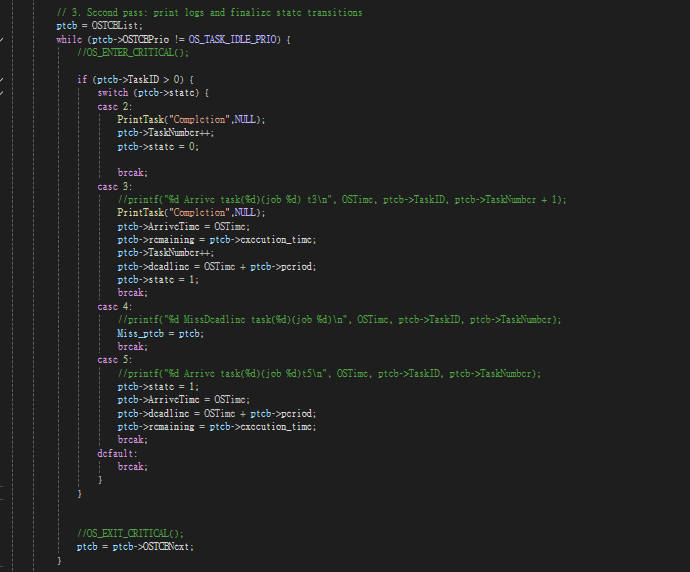
接著會走訪所有任務的 TCB linked list。每個任務會（OSTCBDly--）並檢查是否到達 arrival time。若剛好到達，state 會從 0 轉為 5。state 5 表示任務剛抵達，會在這個 tick 中被排程。同時檢查是否有任務已經超過 deadline，標記為miss(state 4)。



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。

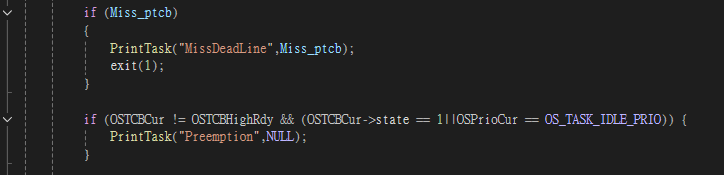


然後第二次走訪 TCB，這時會根據任務狀態印出相對應的 log。例如 state 2 會輸出 Completion 並等待下一次 arrival，state 3 則代表任務完成後立刻產生新 task而進入 running，state 5 也會設定ready。

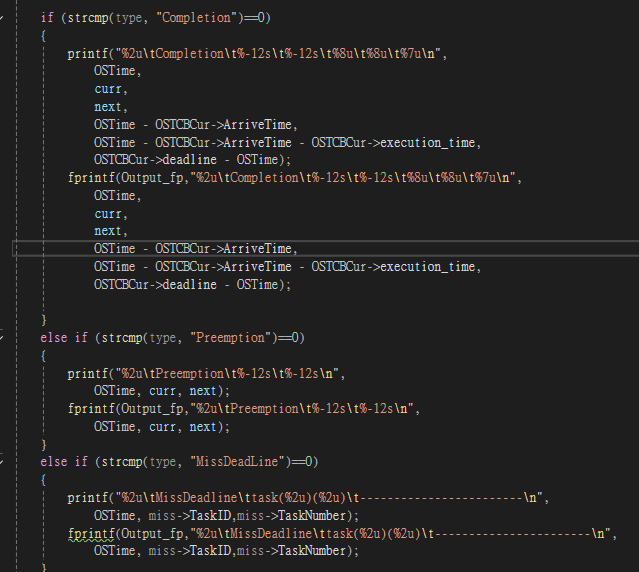
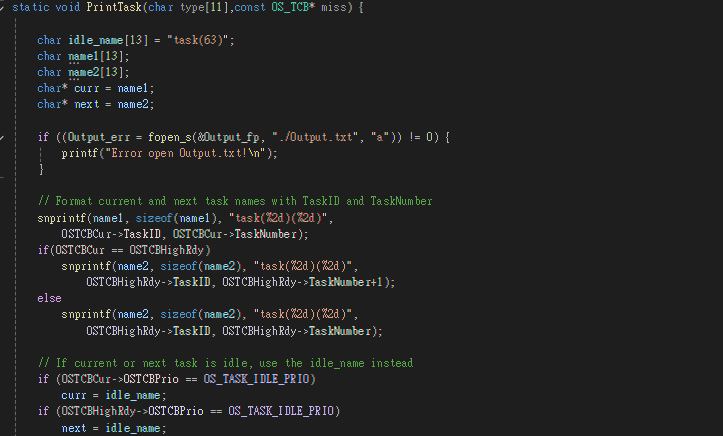


如果Miss，印出Miss的任務並停止系統。

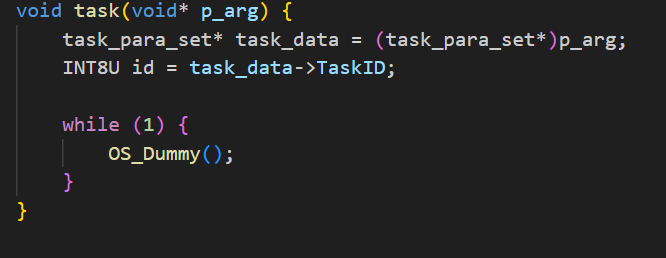
如果新排出的任務與目前執行任務不同，則輸出 Preemption 訊息，並執行 IntExit 與 IntCtxSw 進行任務切換。整體流程會在一個 Tick 中完成所有任務狀態的更新與排程動作。



以下是輸出log用的函數，跟據引數調整輸出內容。



Task本身是無窮迴圈，所以只有tick時才會切換(由timetic控制)



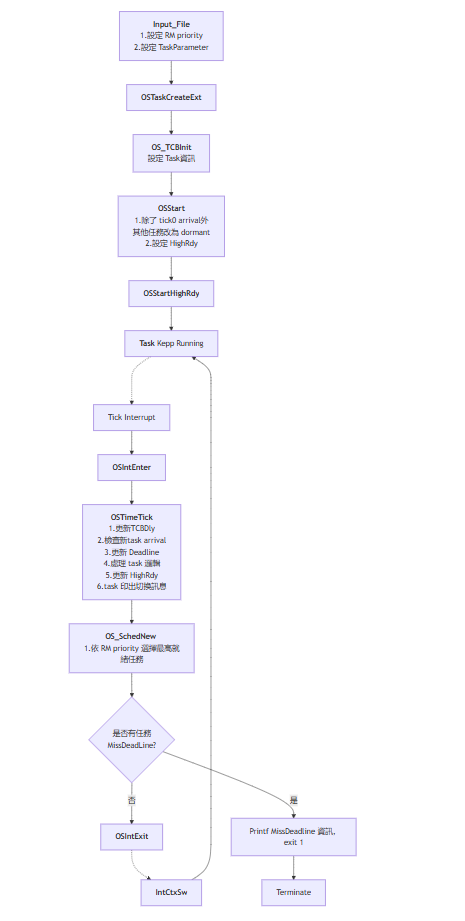
**系統flowchart：**

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。

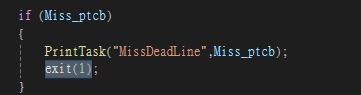
接下來 OSStart 是整個多工系統的啟動點，在這邊我加入了在 tick 0 arrival以外的任務皆先 dormant 的處理，這是為了避免尚未抵達 arrival time 的任務被錯誤排程，同時也設定第一個 ready 任務作為 OSTCBHighRdy，此時第一次呼叫 OSStartHighRdy，開始執行第一個任務。

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)。



1. 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。



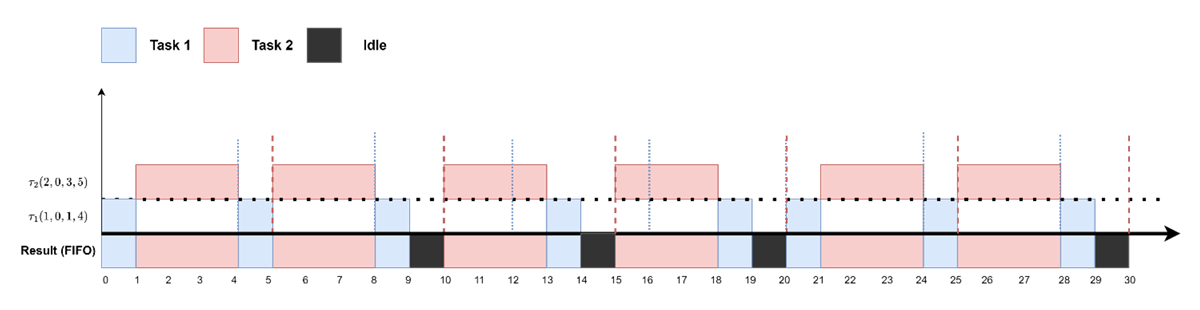
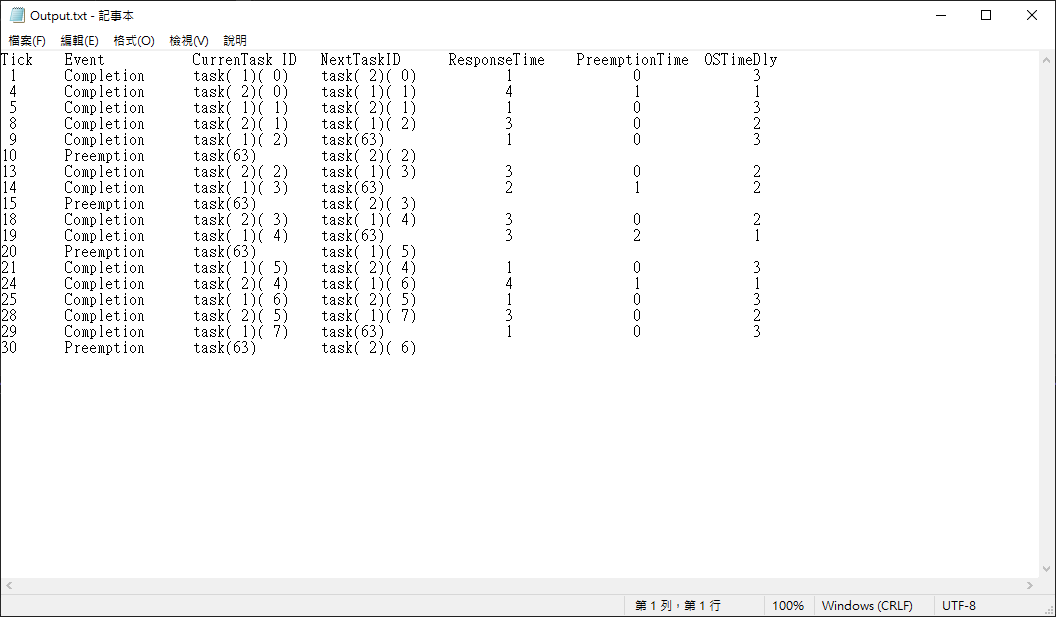
[ PART III ] FIFO Scheduler Implementation [10%]

1. schedule results of examples.

Taskset I

1 0 1 4

2 0 3 5

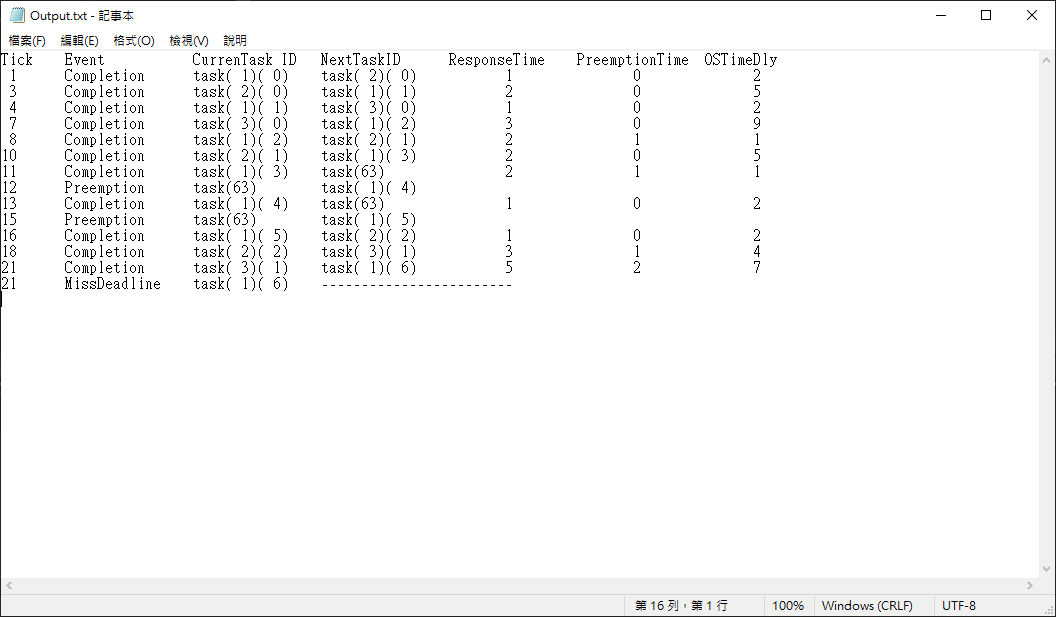


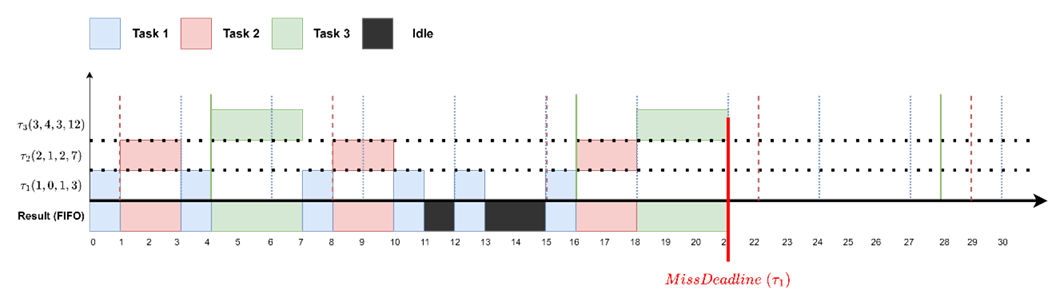
Taskset II

1 0 1 3

2 1 2 7

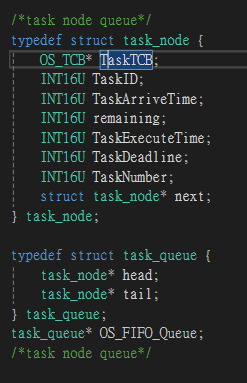
3 4 3 12





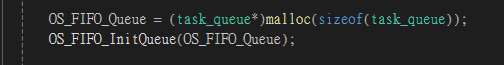
1. Implement (5%)

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

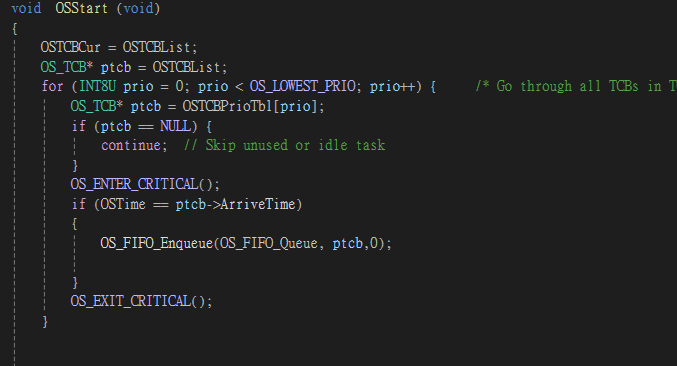


在main.c, main() 創建task\_queue:



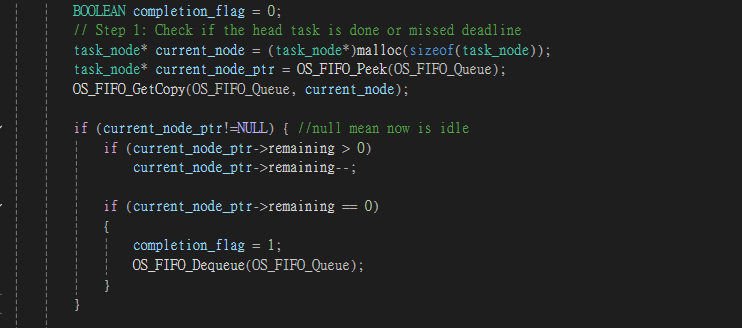


在os\_core.c 的OSStart: 如果任務arrive 放入Queue。

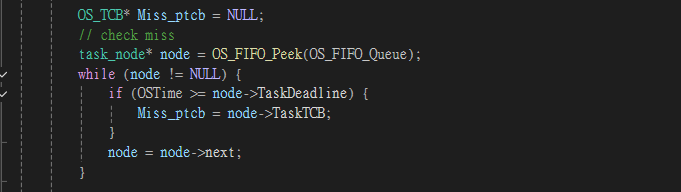


開始運作後，先宣告一個變數 completion\_flag 表示是否有任務在這個 Tick 完成。接下來會從 FIFO 任務佇列中取出目前排在最前面的任務，這段透過 OS\_FIFO\_Peek 取得任務指標，再用 OS\_FIFO\_GetCopy 建立一份 copy 用來記錄完成狀態(因為後面可能dequeue)。如果目前的任務不是 idle 且 remaining 大於零，就將 remaining 減1。如果 remaining 減到零，表示任務執行完畢，設定 completion\_flag 為 true，並從 FIFO中移除該任務（Dequeue）。

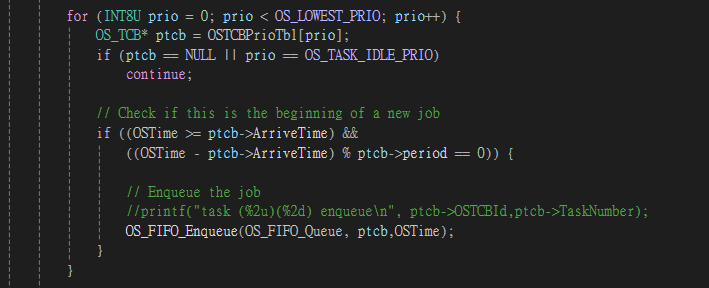




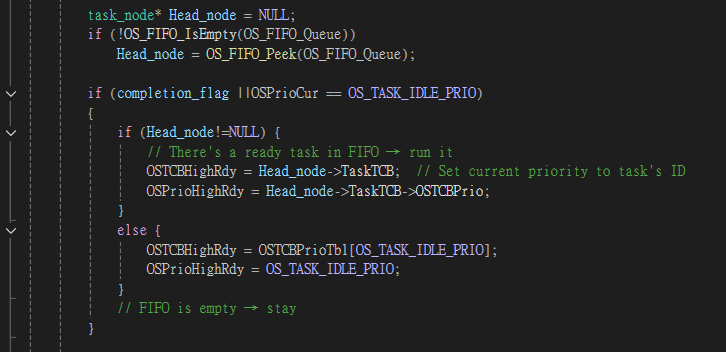
接下來進行 deadline miss 檢查。因為只有要執行的task會在queue裡，所以遍歷整個 FIFO 任務queue，只要有任務的 deadline 小於等於目前的時間，就視為錯過 deadline，將該任務的 TCB 指向 Miss\_ptcb (假設系統抓到一個miss 就紀錄，之後會停止)。



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。

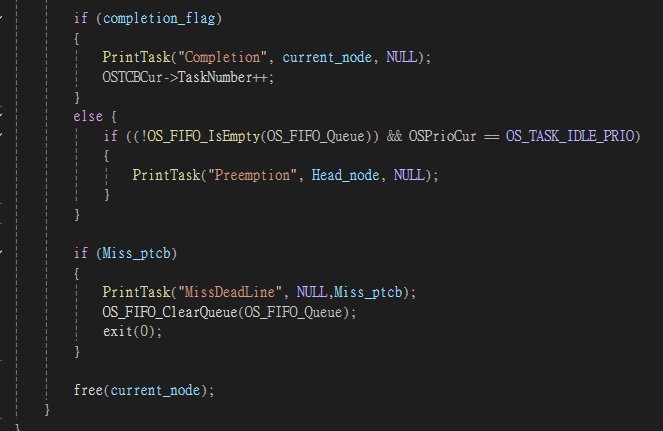


當有任務完成（即 completion\_flag 為 true）或目前執行的任務是 idle 時，表示系統可能需要切換執行中的任務。此時會先檢查 FIFO 任務佇列是否為空：

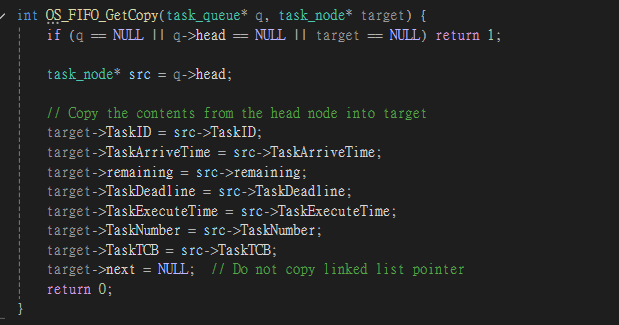
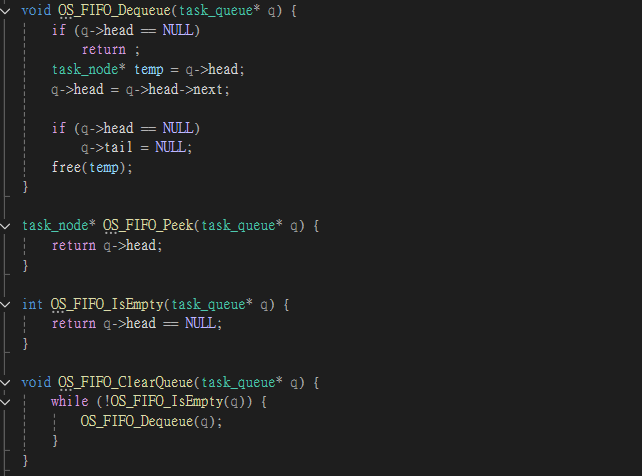
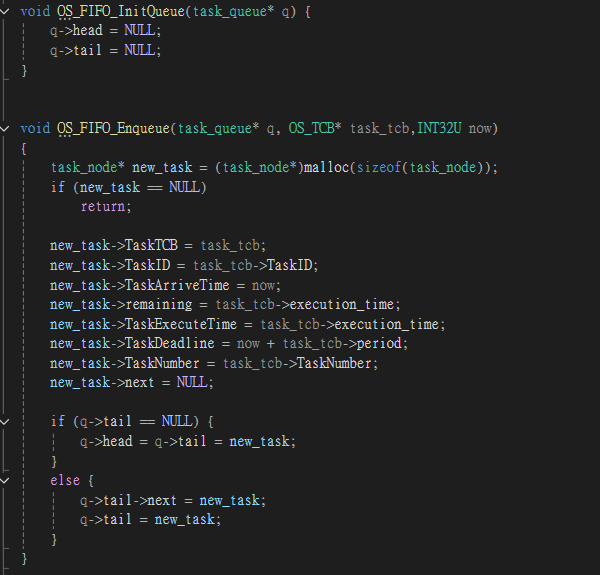
若Queue不為空，代表有其他任務等待執行，則系統會切換至佇列最前面的任務（Head\_node 對應的 TCB）；若Queue為空，則系統會切換至 idle task，讓系統進入閒置狀態直到下一個任務抵達。

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。



以下為Queue 相關函式(放在app\_hook.c)：



Task\_queue的示意圖：

