logo_e_s國 立 台 灣 科 技 大 學

嵌 入 式 作 業 系 統 實 作

指 導 教 授：陳 雅 淑 教 授

**------------------------------------------------------------------------**

嵌入式作業系統實作

Embedded OS Implementation

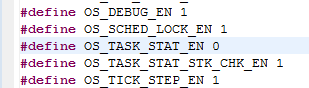
Project 01

班 級 ： 電機碩一

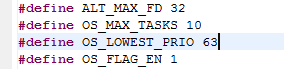
學 生 ： 江昱霖 張晴雯

學 號 ： M10307431 M10407506

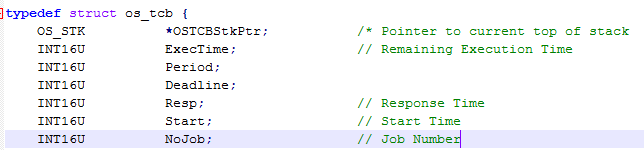
1. **系統環境設定(system.h)**
   * 關閉系統常駐任務Statistic Task (不需執行此任務)



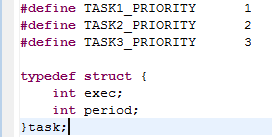
* + 修改為系統最低優先權為63，優先權範圍為0~63



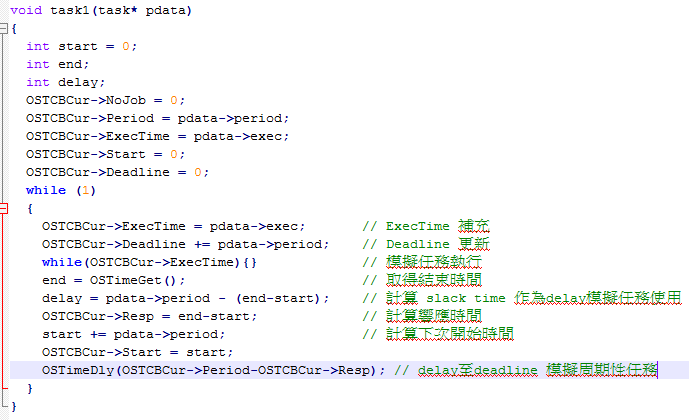
1. **RM (Rate Monotonic Scheduling)**
   * **任務參數創建及初始化設定**
     1. 修改TCB (ucos\_ii.h) ，加入執行時間、週期、截止時間(RM尚未使用到)、響應時間、開始時間以及任務的計數。



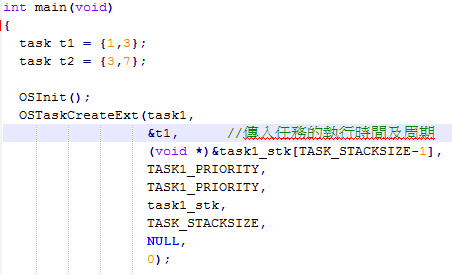
* + 1. 定義參數及創建任務參數資料結構task，作為創建任務的傳入參數。



* + 1. 定義任務函式



* + 1. 創建任務



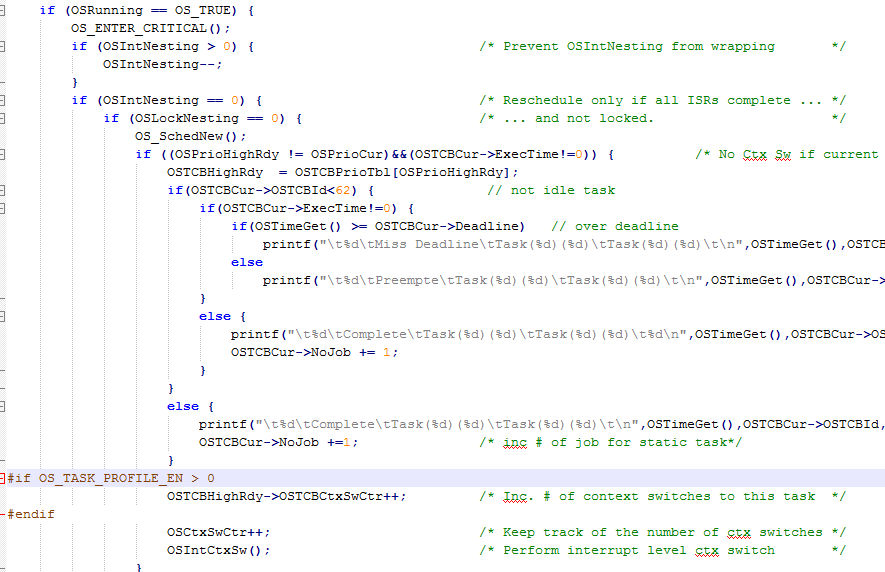
* + **uC/OS ii Kernel 修改**

在原本的系統下，我們使用週期作為優先權定義的依據，則其就是RM的排程。在每個time tick對目前的任務執行時間作遞減，模擬精確的任務執行。當time tick觸發時，為一中斷事件，會確認各個任務當前的狀況，並離開中斷時會呼叫OSIntExit()，OSIntExit()內又會呼叫OS\_Sched()，檢查是否有更高優先權的任務與任務切換，因此在這兩個函式中來顯示任務的執行情況。

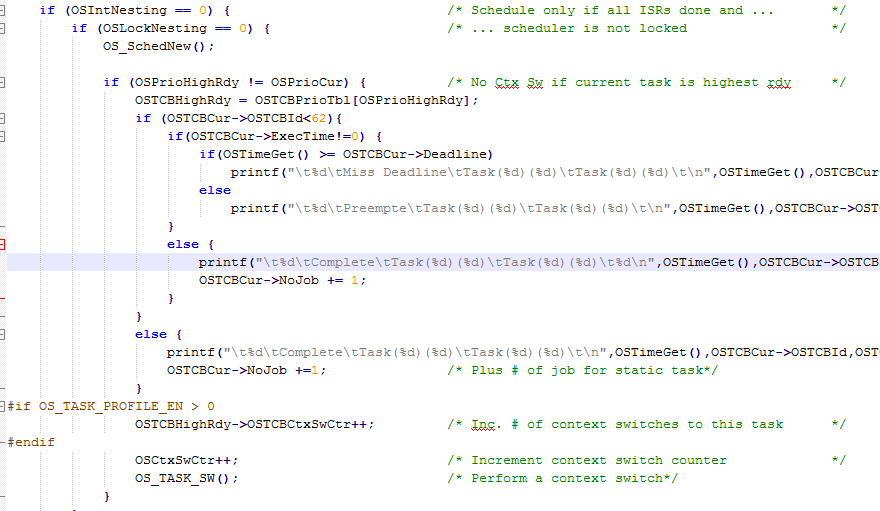
* + 1. 模擬執行時間 (OS\_TimeTick)



* + 1. 離開中斷部分顯示任務資訊 (OSIntExit)

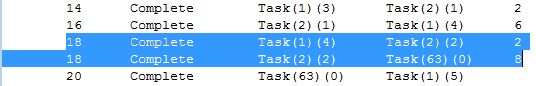


* + 1. Scheduler顯示任務資訊 (OS\_Sched)，與OSIntExit增加部分大致相同。



* + **問題與討論**

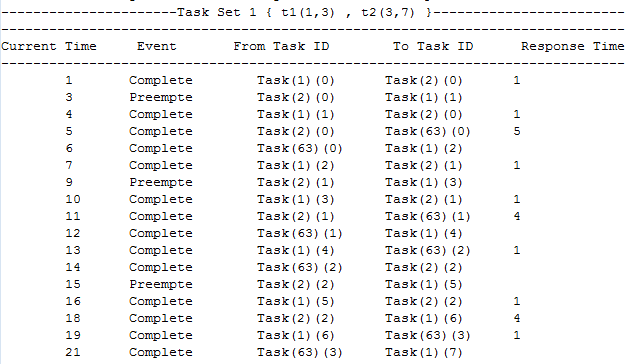
由於是利用ExecTime來模擬任務執行，雖然當下ExecTime以歸零視為任務完成，但其實還有設定Delay的部分才算整個任務完成，當TimeTick=16時，因為Task2還未完成Delay的設定而被Task1搶占，導致後面排程的出錯。因此必須在OSIntExit處理雖執行時間為零卻被搶佔導致錯誤的問題。

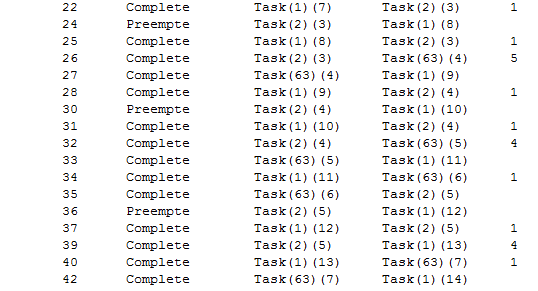


利用一判斷式來阻擋在執行時間為零且還沒完成Delay設定而被搶佔，則當執行時間為零時，不會進入作任務置換，可以順利執行設定Delay，完成整個任務。

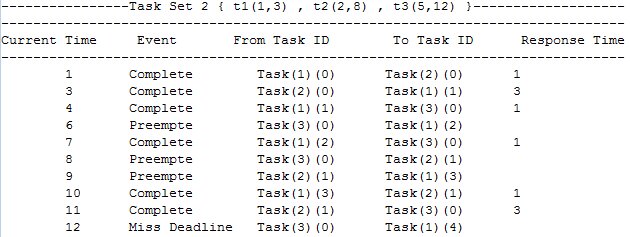


* + **結果(Time Tick 0 - 42)**
    1. Task set 1 模擬:





* + 1. Task set 2 模擬:



* + **Schedulability Analysis**
    1. RMA

The highest priority tasks have the smallest periods. Priorities are assigned off-line.

 ， = response time ， = 每個task 的執行時間

==＞  ， = 每個task 的 period

Task Set 1 ：τ1(1,3)、τ2(3,7)

T1：

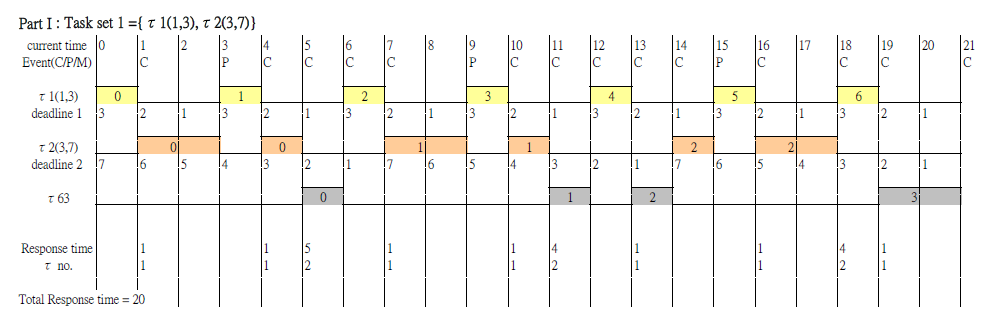
 ≦ 3

T2：

 ≦ 7

 ≦ 7

 ≦ 7 ， ok



Task Set 2 ：τ1(1,3)、τ2(2,8)、τ3(5,12)

T1：

 ≦ 3 ， ok

T2：

 ≦ 8， ok

 ≦ 8 ， ok

T3：

 ≦ 12

 ≦ 12

 > 12

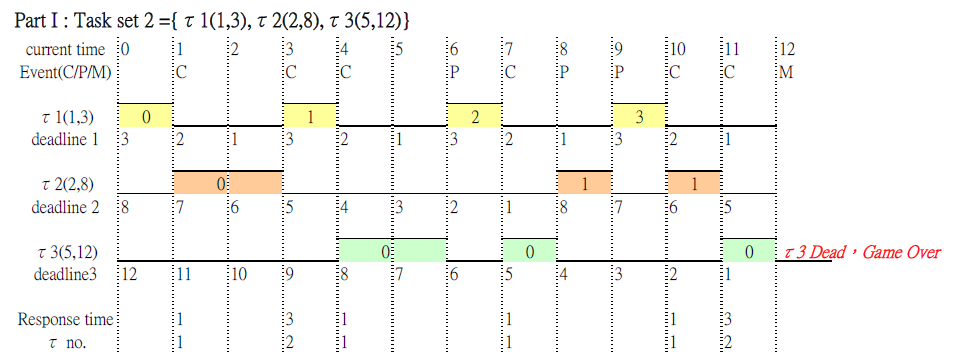
==＞ the task set failed

而 Utilization factor U =  ≦ U(n) = n

Task Set 1 ：τ1(1,3)、τ2(3,7)

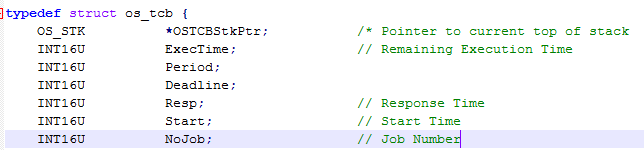
U =  ≦ U(2) = 0.828

Task Set 2 ：τ1(1,3)、τ2(2,8)、τ3(5,12)

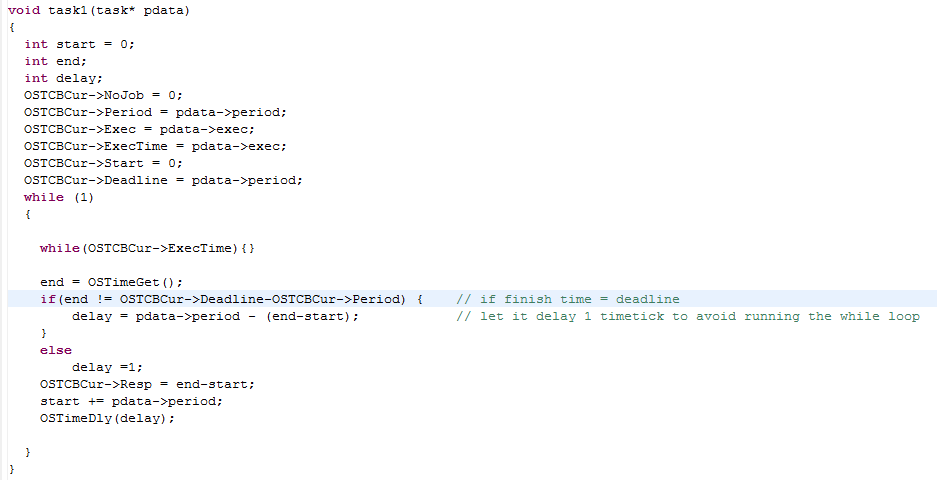
 U =  > U(3) = 0.779

==＞ 不一定能排入，由前RMA分析結果已知不能被排程。

1. **EDF (Earliest Deadline First Scheduling)**
   * **任務參數創建及初始化設定**
     1. EDF係使用截止時間作為動態優先權的設定依據，因此在RM的基礎上多增加了截止時間於TCB中(ucos\_ii.h)。



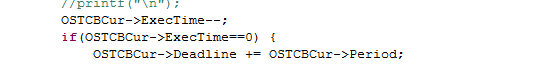
* + 1. 定義任務函式



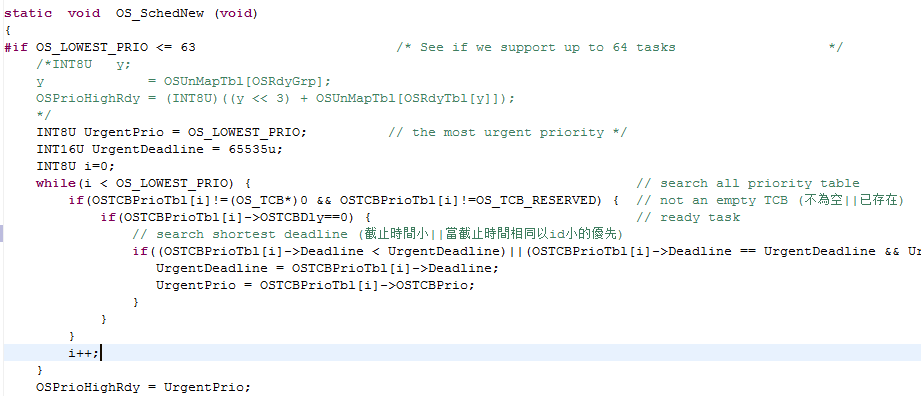
* + **uC/OS ii Kernel 修改**

EDF以截止時間定義優先權，是一動態優先權的排程方式。因此在任務有異動(加入/完成)時就必須判斷目前哪個任務的截止時間最早，其優先權為最高，因此我們在Time Tick中判斷當ExecTime歸零時代表該任務即將完成，並更新其截止時間，再讓Scheduler去判斷最高優先權。在Scheduler部分摒棄了原查表的方式，利用一迴圈尋找存在的任務中截止時間最早的任務為優先權最高的任務，使其能夠馬上被執行。

* + 1. 任務截止時間更新 (OS\_TimeTick)



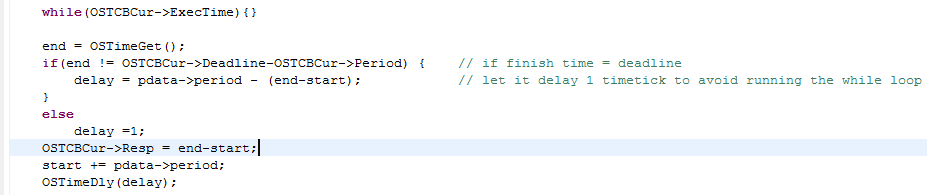
* + 1. 修改Scheduler尋找最高優先權的部分，改以最早截止時間為優先(OS\_SchedNew )



* + **問題與討論**

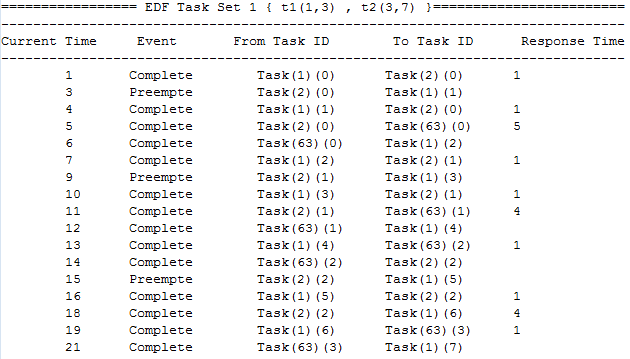
在total utilization =1的情況下，在hyper period的時間點，每個任務的deadline均相同，當下一定是最低優先權的任務完成，但由於完成的時間剛好壓在截止時間上，該任務的Delay = 0，會直接重新執行while迴圈，導致一些參數的重複計算設定而出錯，後續的排程也為之大亂。

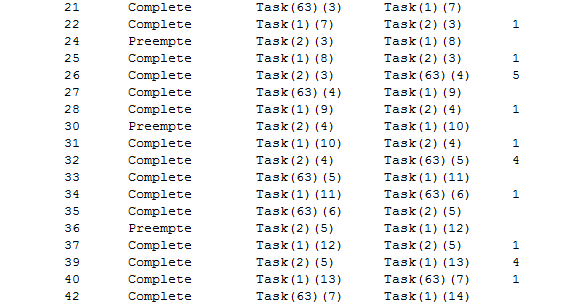
因此我們得知根本原因是會重複執行while，所以當在這種狀況下強制使其Delay一個TimeTick，使其在該時間點結束能夠保持正確的參數設定。因為這種狀況一定發生在hyper period上，而執行的任務一定是最低優先權的任務，接著執行的一定是最高優先權的任務，因此強制使最低優先權任務延遲一個TimeTick對整個系統排程是沒有影響的。



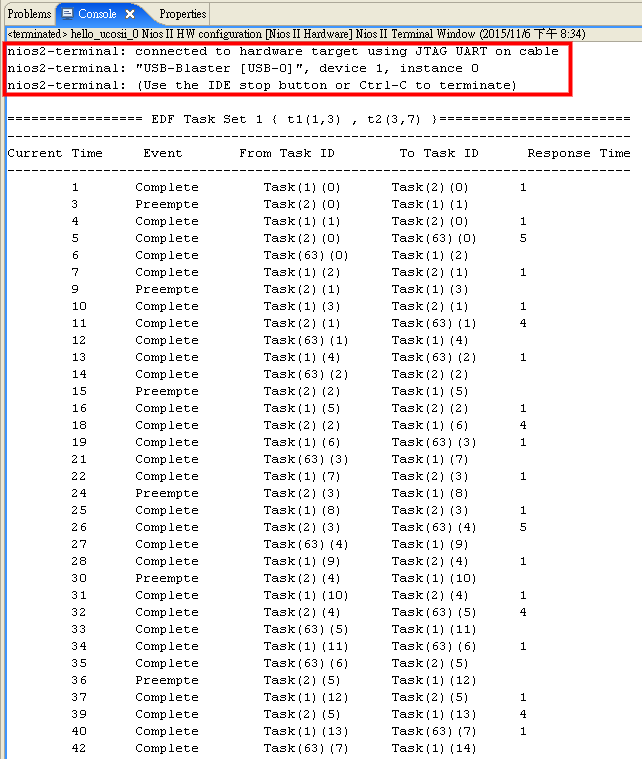
由於是搜尋每個任務找尋最早截止時間的任務，因此其時間複雜度為O(n)，而原本kernel的時間複雜度為O(1)，但其是以空間換取時間，利用建表的方式來快速找尋最高優先權。

* + **結果**
    1. Task set 1 :
* 模擬

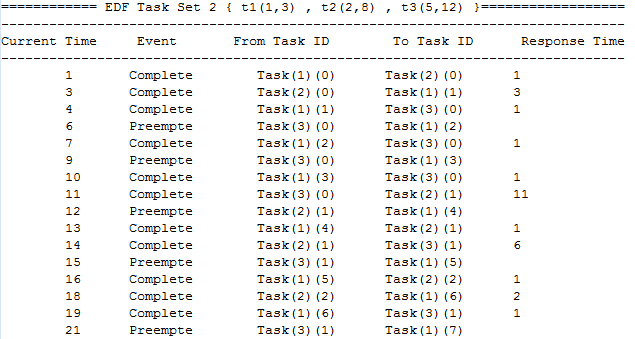


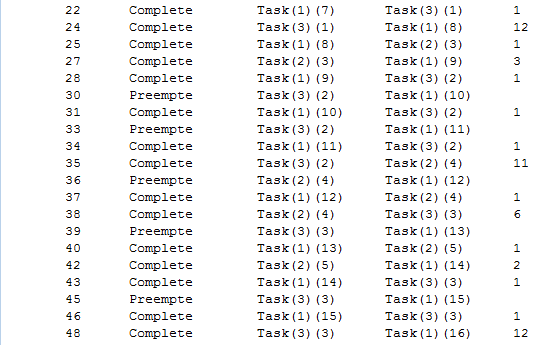


* 硬體上運行

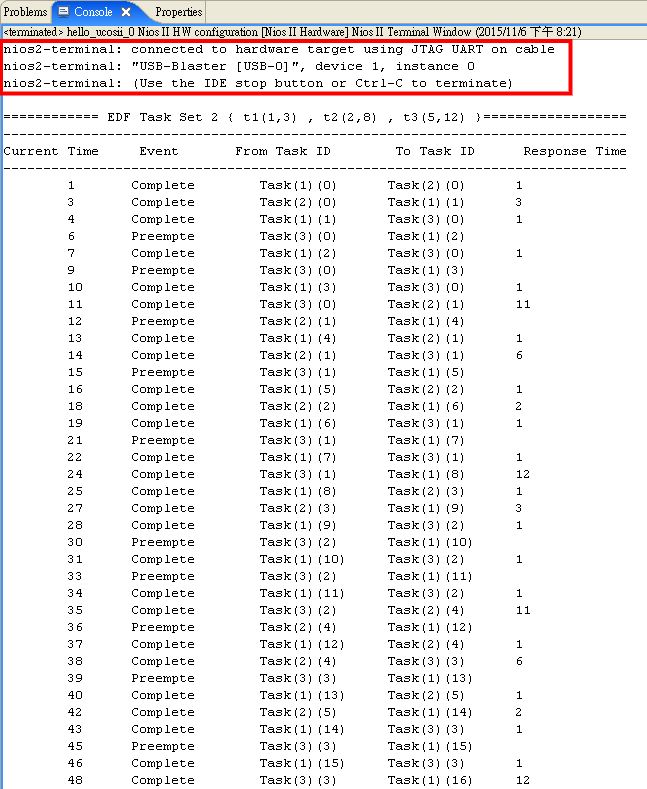


* + 1. Task set 2 :
* 模擬





* 硬體上運行



* **Schedulability Analysis**

A job having an earlier deadline is assigned to a higher priority.

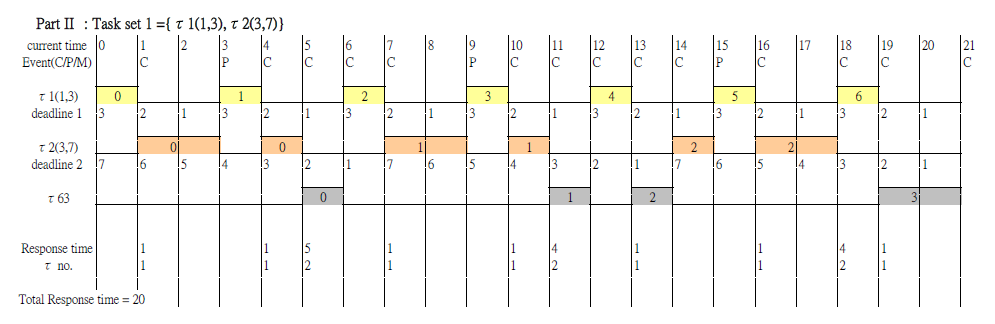
A set of tasks is schedulable by EDF if and only if its total CPU utilization is no more than 1.

若每個task 工作時間都 < T 而加起來 U>1 則不能排!

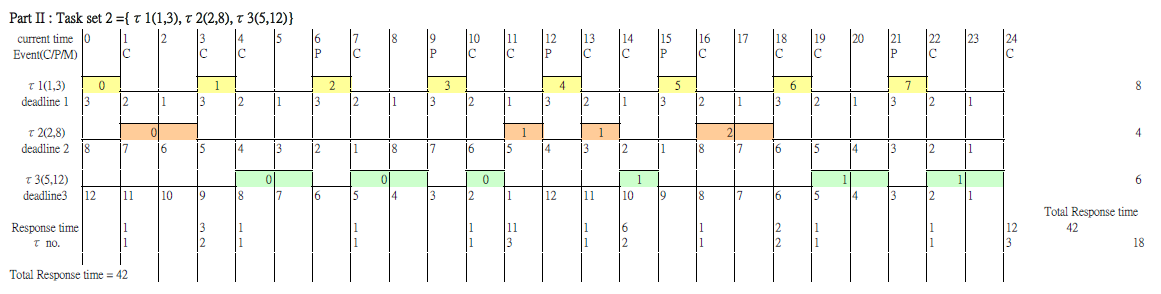
即 U =  ≦ 1

Task Set 1 ：τ1(1,3)、τ2(3,7)

U =  ≦ 1 ==＞ 可排程



Task Set 2 ：τ1(1,3)、τ2(2,8)、τ3(5,12)

 U =  ≦ 1 ==＞ 可排程

1. **LSTR (Least Slack Time Rate Scheduling)**
   * **構想:**

事有緩急輕重，現實生活中對於事情的安排我們可能會考慮最急迫及最重要的為優先，因此我們構想出能夠兼具事情急迫性與重要性的演算法，初步稱之為最大壓力排程法，剩餘工作量與剩餘可執行時間的比例為之壓力:

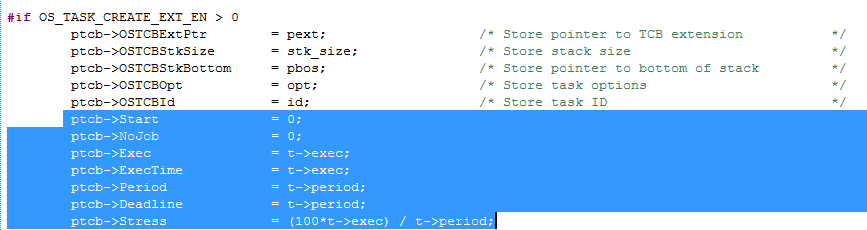
以壓力大的任務為優先，在每個時間點都會檢查所有任務的壓力，動態調整優先權，與EDF最大的不同為EDF只考慮截止時間，不能確認每個當下刻不容緩的任務，一旦EDF執行高優先權時有一段時間其他任務不被考慮，因此我們的方式能夠確認每個當下最急迫的任務進行排程，其utilization bound也與EDF相同。

後來開始著手尋找相關演算法確認是否已被提出，結果找到了一篇2010年提出與我們構想一模一樣的演算法叫做LSTR(Least Slack Time Rate Scheduling)，其是在EDF及LST的基礎下發展而來，運行在多核的環境下，因每個時間點都去檢查任務狀況，雖然多花了點時間去做檢查任務狀態，但能夠減少其他處理器閒置的情形。

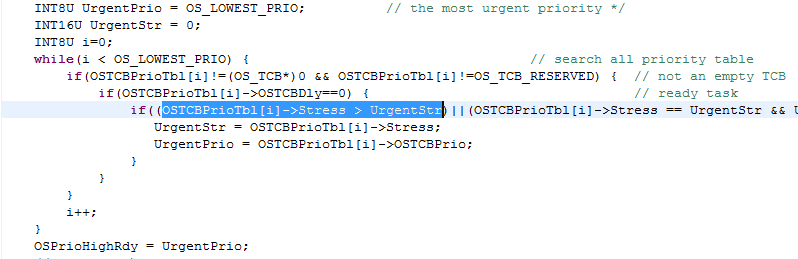
* + **任務參數創建及初始化設定**
    1. 此方法係使用壓力作為動態優先權的設定依據，因此在EDF的基礎上多增加了壓力於TCB中(ucos\_ii.h)。



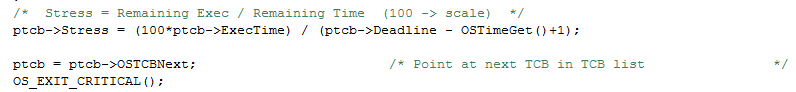
* + 1. 我們將所有任務參數初始化都放進了TCB初始化的函式中(TCBinit)，由於使用整數計算，因此在**計算Stress時乘上100來修正小數的情形**。



* + **uC/OS ii Kernel 修改**
    1. Scheduler與EDF類似，只是改以尋找壓力最大的任務進行排程。



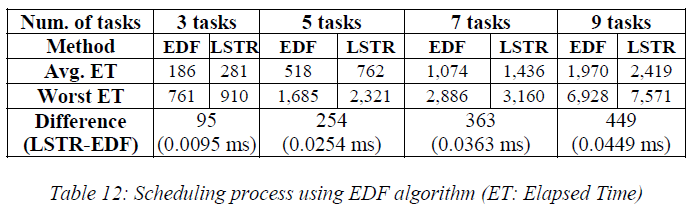
* + 1. 每個任務的壓力在每個TimeTick均會更新 (OS\_TimeTick)



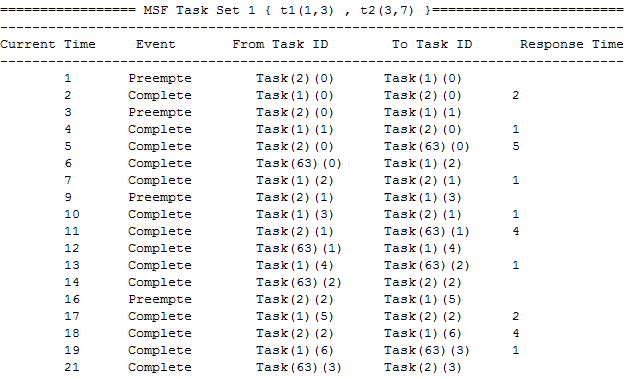
* + **問題與討論**

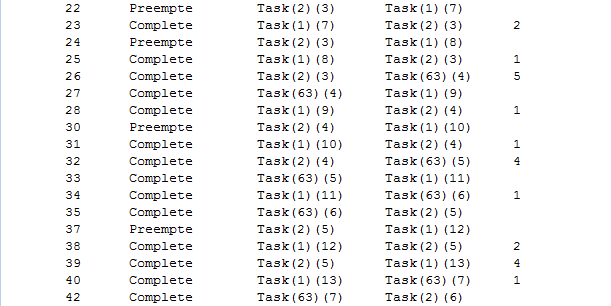
我們的構想已存在於2010 年JCUS論文中[1]，此論文是探討在多核的環境下，LSTR相較EDF能夠大幅減少其他核心閒置的情形，且可排程性與EDF相同，能夠在total utilization小於1的情形下排程，具有與EDF相似的特性。

其缺點就是必須在每個時間點多花一些時間計算每個任務的執行速率，但也因為這樣能夠及時知道哪個任務是刻不容緩的，從實驗數據中能夠看出當task越多，其overhead越大，為O(n)線性成長的趨勢。

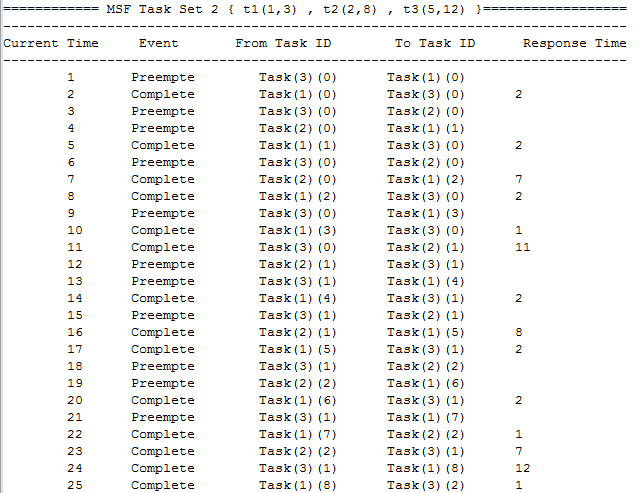


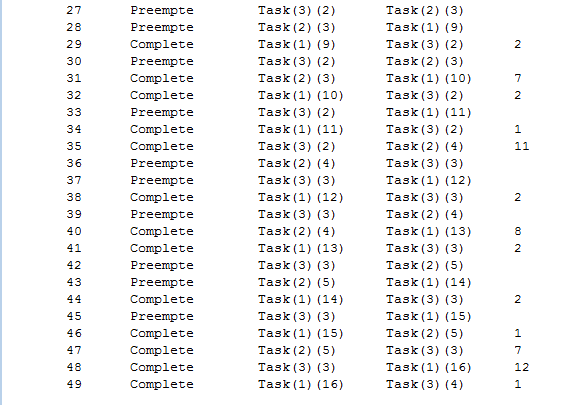
* + **結果**
    1. Task set 1 模擬:





* + 1. Task set 2 模擬:





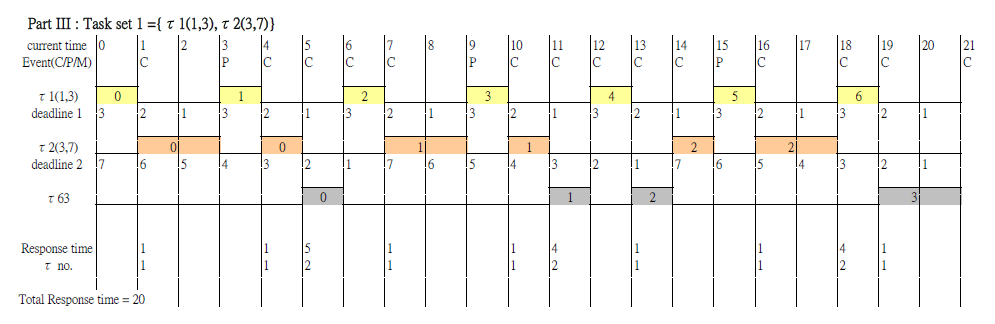
* **Schedulability Analysis**

一個Task是否可以排入，同EDF : A set of tasks is schedulable by EDF if and only if its total CPU utilization is no more than 1.

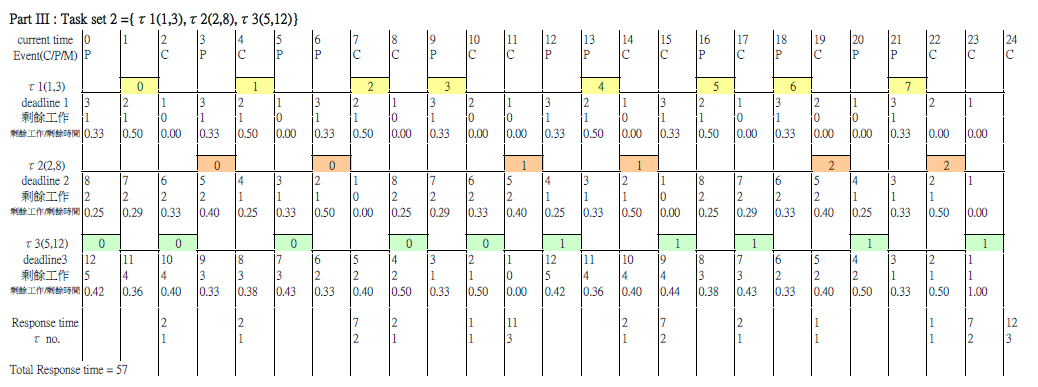
若每個task 工作時間都 < T 而加起來 U>1 則不能排!

即 U =  ≦ 1

Task Set 1 ：τ1(1,3)、τ2(3,7)

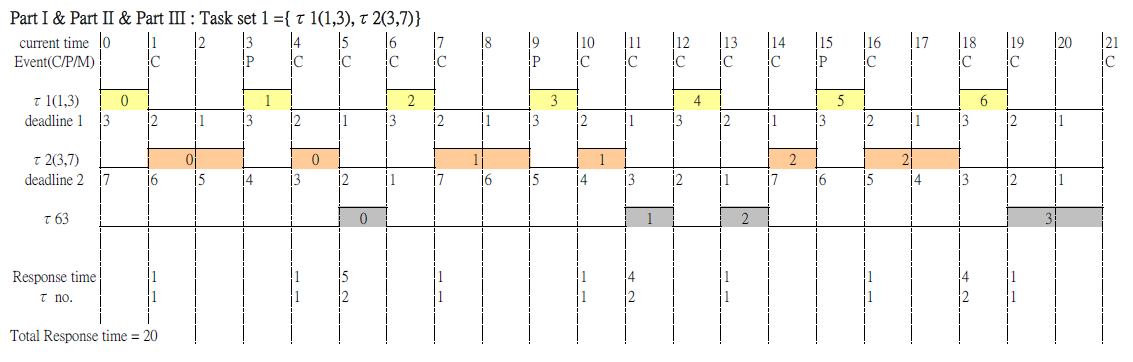
 U =  ≦ 1 ==＞ 可排程

Task Set 2 ：τ1(1,3)、τ2(2,8)、τ3(5,12)

 U =  ≦ 1 ==＞ 可排程

1. **心得**

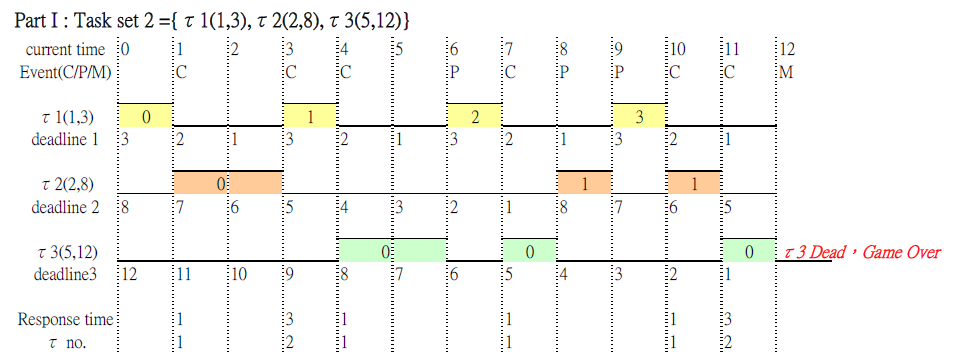
本次project最難的地方是第一部分，不是因為要做的事情難，而是初次接觸這個OS，對其不熟悉，需要花很多的時間去trace，也必須了解整個系統流程，才能夠知道需要修改甚麼地方、要在哪裡得到所需的資訊以及如何找出問題發生的位置，這些都花了好一段時間才逐漸地掌握，通盤了解系統流程後，後續的部分也不是甚麼大問題了。最有趣的是第三部分，藉由小組的方式能夠互相討論，每個人所看到的問題點都不同，更能夠激發出一些創意的想法來設計我們自己的排程演算法。藉由尋找相關學術論文及資訊，也能夠認識更多不同用途的演算法，甚至發現我們的演算法在幾年前剛剛被發表出來，證明我們也是有能力想出一個不錯的排程演算法。本次project中取RM、EDF、LSTR三種不同的排程法進行工作排序，並將比較三種不同排程的差異。



由Task Set 1對三種不同排程法的測試結果發現，三種不同的排程法所得到的排程結果相同 : 每21個時間單位即可完成一次循環，其中可執行7個τ1以及3個τ2，並且將τ1與τ2分成13個小區塊完成，其中執行了16次Context Switch(考慮常態任務轉換及任務搶佔)且Total Response Time=20。

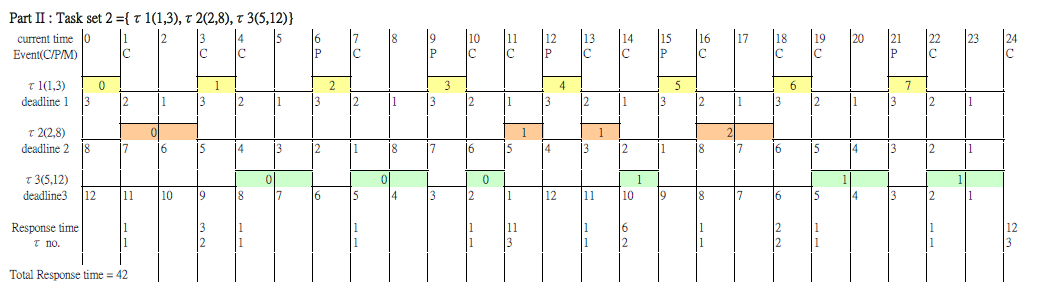
而Task Set 2的情況則可以分別出三種排程法不同的效能。

首先是RM:



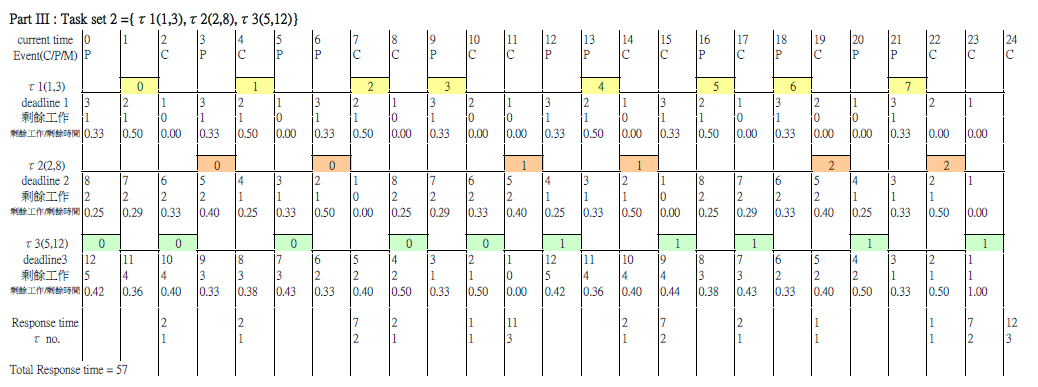
在current time=12時miss deadline，從計算中也可以看出此種排程法無法將工作排入。

EDF:



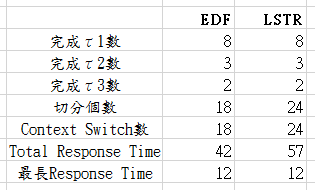
每24個時間單位即可完成一次循環，其中可執行8個τ1、3個τ2以及2個τ3，並且將工作分成18個小區塊完成(分別為8個τ1片段、4個τ2片段以及6個τ3)，其中執行了18次Context Switch且Total Response Time=42，最長的Response Time為12。

LSTR:



每24個時間單位即可完成一次循環，其中可執行8個τ1、3個τ2以及2個τ3，並且將工作分成24個小區塊完成(分別為8個τ1片段、6個τ2片段以及10個τ3)，其中執行了24次Context Switch且Total Response Time=57，最長的Response Time為12。

從以上結果可以整理出此表格 :



由此表可發覺EDF是較有效率排程方法，然而LSTR則可兼顧工作的重要性與事情急迫性且LSTR可以讓每一個Task的進度都隨著時間的經過，而有均勻的進展。

1. **Reference**

[1] Myunggwon Hwang, Dongjin Choi, Pankoo Kim, ” **Least Slack Time Rate First: an Efficient Scheduling Algorithm for Pervasive Computing Environment**”, Journal of Universal Computer Science, 2010