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

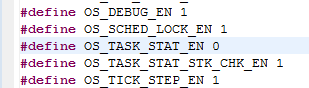
Project 03

班 級 ： 電機碩一

學 生 ： 江昱霖

學 號 ： M10307431

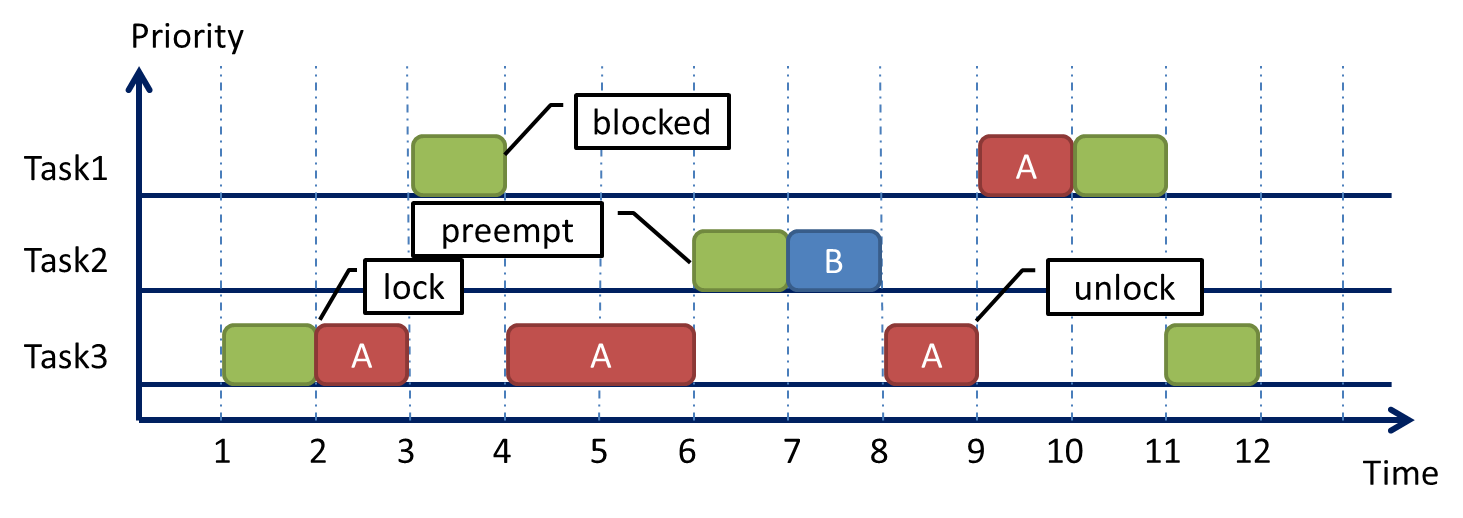
1. **系統環境設定(system.h)**
   * 關閉系統常駐任務Statistic Task (不需執行此任務)



1. **SRP (Stack Resource Policy) Implement**

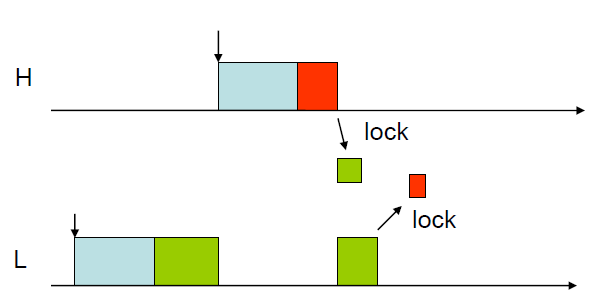
由於系統採用最早截止時間優先演算法EDF(Earliest Deadline First)是非常著名的實時排程演算法，一種優先搶占式的排程法，能夠保證當前運行最高優先權的任務。然而在實際運用上，任務間的資源共享，佔有資源往往會造成低優先權任務長時間佔有或阻塞高優先任務，高優先權任務反而需要等待低優先權任務執行完畢，這可能會造成系統的不確定性甚至崩潰，這即是優先權反轉的問題。

* + **Priority Inversion**

****

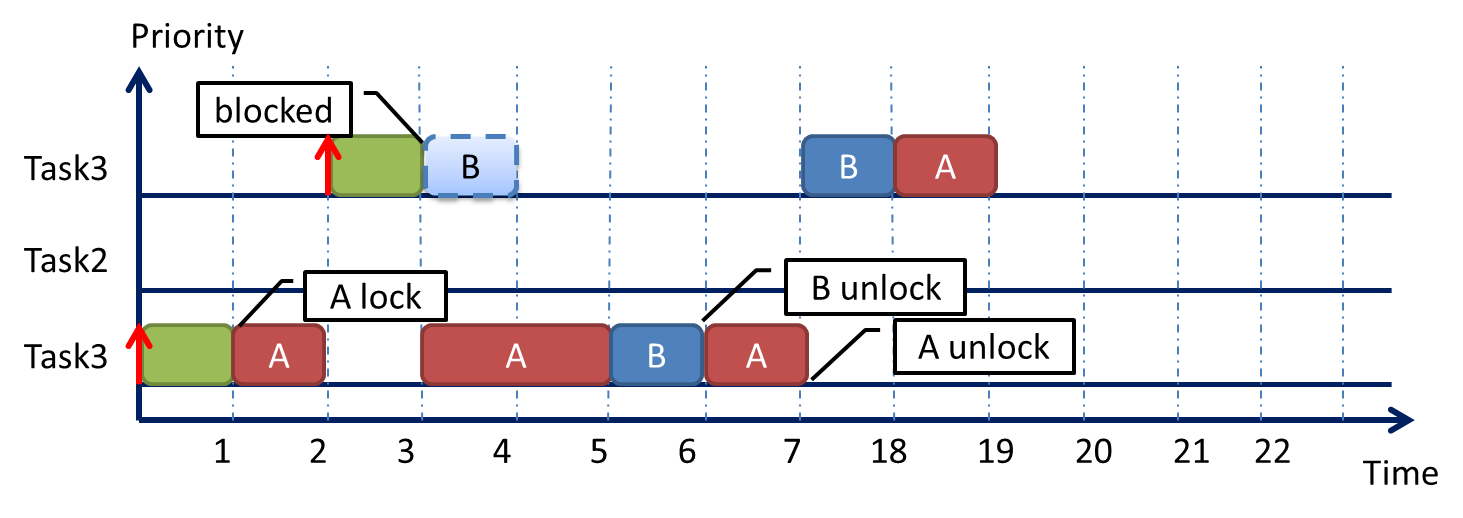
為解決反轉問題，發展出了PIP(Priority Inherit Protocol)，當有高優先權也需要取得資源時，將持有資源的低優先權任務去繼承高優先權任務的優先權，避免中優先權任務搶佔的可能性，減少阻塞等待的時間。但此方式無法解決Deadlock的問題。當雙方任務分別持有資源尚未釋放，又同時需要取得對方資源時，就會陷入雙方互相等待對方釋放資源的死胡同，這就是Deadlock問題。

* **Deadlock**



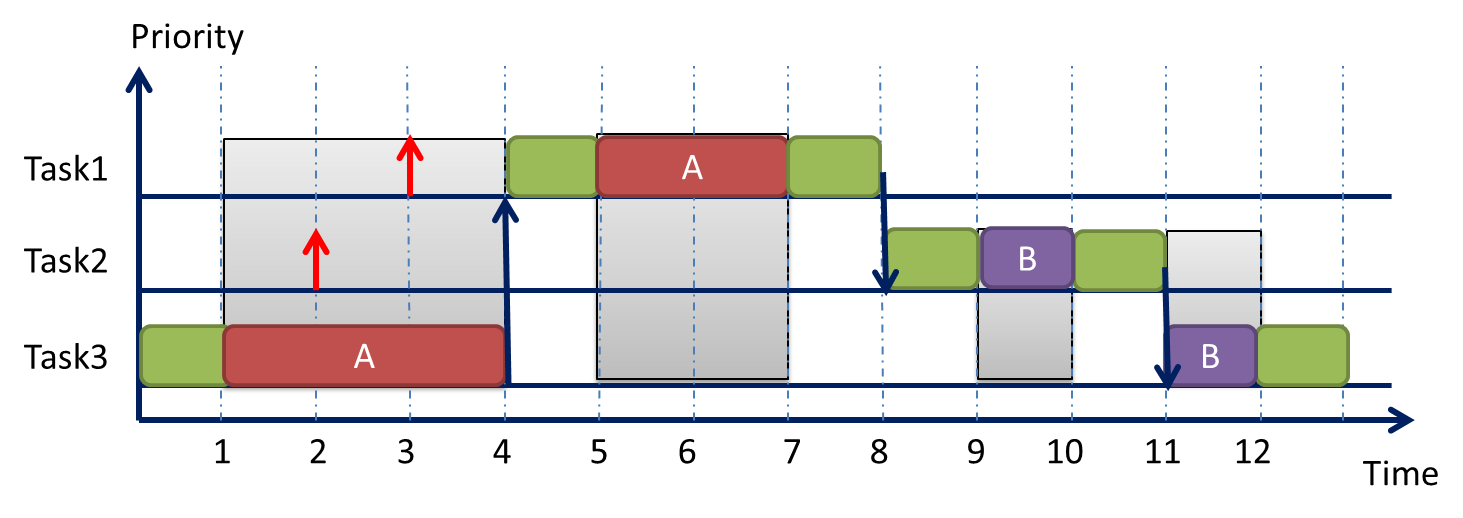
因此進一步發展出了PCP(Priority Ceiling Protocol)，當任務在拿取資源時需確認是否能夠一次拿取全部的所需資源，否則就可能會陷入deadlock的情況，每個資源會依據被任務存取的優先權高低訂定出Ceiling，用來保證任務存取的優先性，避免多個任務分別持有又互相等待的情況。

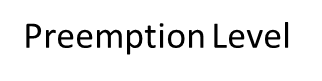
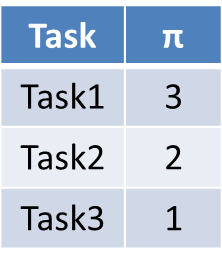
* **PCP**

****

由於PCP是以任務的優先權來訂定Ceiling，因此適用在固定優先權的RM上，而動態優先權的EDF則無法適用。所以發展出了SRP (Stack Resource Policy)，根據任務的週期訂定每個任務的Preemption Level，其代表的意義就如同PCP中的Priority一樣，任務對於資源存取的優先順序，只是在EDF中任務的優先權會依據時間不同而改變，所以PCP才無法適用，SRP則是依據持有資源任務的Preemption Level來訂定系統的Ceiling，確保在存取資源時能夠一次取得所需資源而不被他人阻塞甚至造成鎖死。且PCP是在存取資源時檢查Ceiling與自身權限的關係，而SRP則是在任務開始執行時就檢查權限，如此一個任務至多就是2次的Context Switch，優化了任務在存取資源時才發現權限不足需要切換任務的情況，能夠省去許多不必要的任務切換。

* SRP



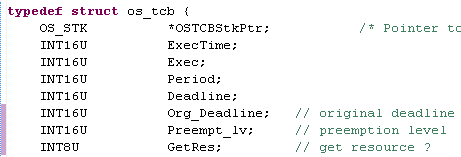


為實作SRP於uCOS II上，我們可以使用Mutex這個功能來做資源存取的保護與控管，由於uCOS II不支援多個任務同時擁有相同優先權，因此我們必須做繼承優先權的定義與管理。Mutex本身即具有優先權繼承的功能，當某一資源已被低優先權持有，且有高優先權也需存取此資源時，Mutex會將低優先權任務的優先權繼承為使用者定義的優先權級別，達到優先權繼承的概念，確保任務不會被中斷並保護資源的使用。

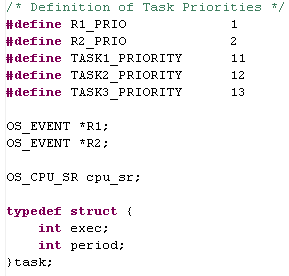
* + **任務參數創建及初始化設定**

由於SRP需使用到繼承的部分，而系統是建立在EDF排程下，所以需要將原始的截止時間記錄起來，以便在釋放資源時回復原優先權的使用。且加入了Preemption Level用來定義該任務對資源的權限。因此在TCB下加入截止時間與SRP的相關參數及修改函式達到任務的設定及初始化。

* + 1. 修改TCB (ucos\_ii.h) ，加入執行時間、週期、截止時間、原始截止時間、搶占等級以及持有資源旗標，用來計算當前任務狀態及顯示的相關資訊。



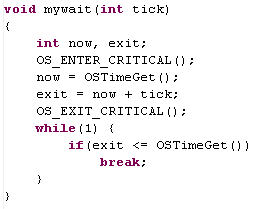
* + 1. 定義優先權參數及創建任務參數資料結構task，作為創建任務的傳入參數，用來初始化任務的設定，並建立資源的事件結構。



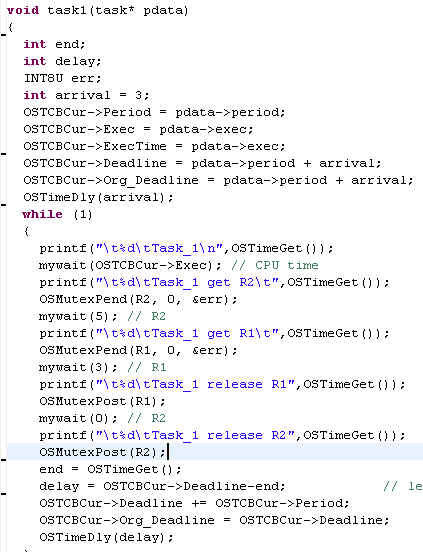
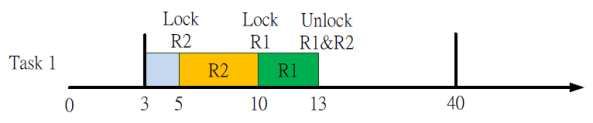
Task Struct

Resource Event

* + 1. 定義執行時間函式，用來模擬任務執行。



* + 1. 定義任務函式，以無限迴圈來表示週期性的任務，並利用先前建立的等待函式來模擬任務的執行時間，使用Mutex來實現任務間共享資源的控管，用delay函式來模擬任務等待下個週期的到達。



釋放資源

拿取資源

到達時間設定

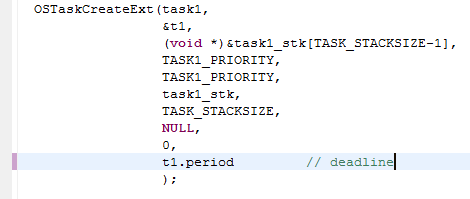
任務參數初始化

任務執行迴圈

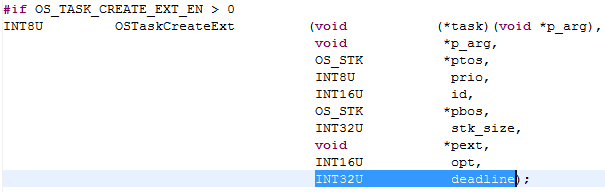
* + 1. 創建Mutex來保護資源，並傳入該資源被存取的優先權限。

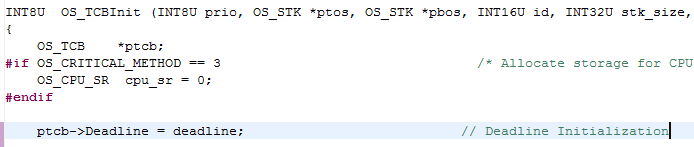


* + 1. 創建任務，傳入先前加入的任務資料結構，並修改該函式傳入截止時間做為初始化設定。



修改創建任務函式(ucos ii.h、os\_task.c)，並在OS\_TCBinit(os\_core.c)中做截止時間的初始。

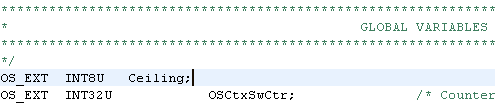


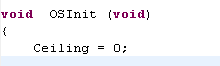


* + **uC/OS ii Kernel 修改**

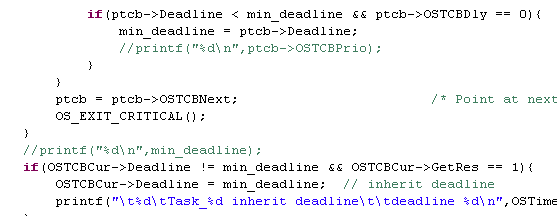
基於EDF排程的系統中，SRP會依據Preemption Level與Ceiling來控管資源的的使用，優先權繼承的部分Mutex已經有提供基本的操作了，還有截止時間的繼承是我們所需要實作的。因此在任務有異動(加入/完成)時就必須判斷目前的就緒任務中是否有截止時間較早的任務需要被繼承其截止時間。必須OSMutexPend以及OSMutexPost中來調整Ceiling，並在Scheduler中加入Ceiling的判斷，以便正確的調度任務。

* + 1. 首先宣告全域變數Ceiling，反映目前資源使用的優先權高低。(ucosii.h)，並在系統初始時設定為0。



(os\_core.c)

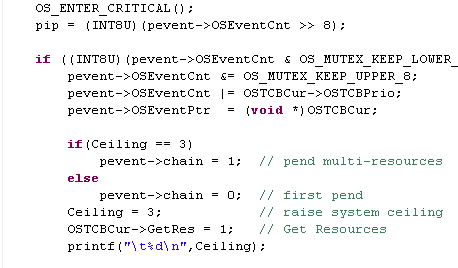
* + 1. 在EDF排程下，需要隨時判斷是否有高優先權的工作就緒，為防止被搶占須完成截止時間繼承的步驟，使低優先權任務能夠完整完成資源的使用，並不被搶占，達到SRP的基本目的。在TimeTick中去判斷是否有截止時間較早的工作就緒，若有則繼承其截止時間，使其在EDF排程下達到SRP不被搶佔的目的。



Deadline繼承

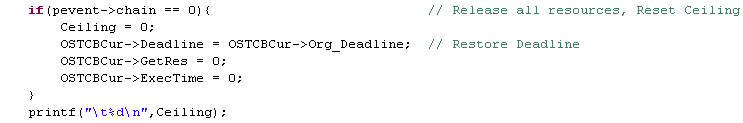
搜尋較短deadline

* + 1. 在MutexPend時，將Ceiling提升為3 (最高為3)，並判斷是否持有多個資源，並將其記錄在ECB中，若是持有多個資源意味在Post時，只有在釋放最後一個資源時才能將Ceiling降為0，其他都將維持3表示手上仍擁有資源尚未釋放。(os\_mutex.c)



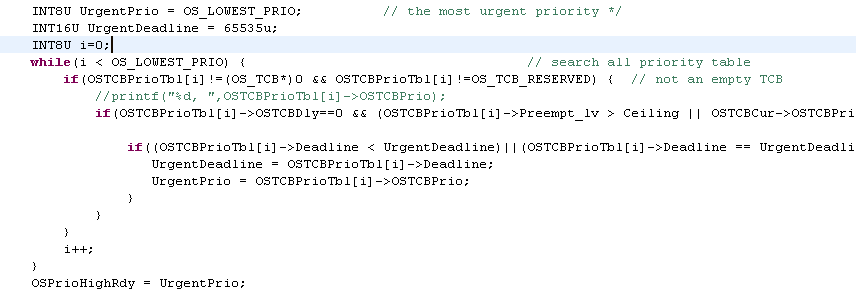
持有多資源

* + 1. 在MutexPost時，依多資源持有情況來決定是否要將Ceiling降為0，因先前的截止時間繼承關係，釋放資源時須將截止時間還原，並更新資源持有的情況。



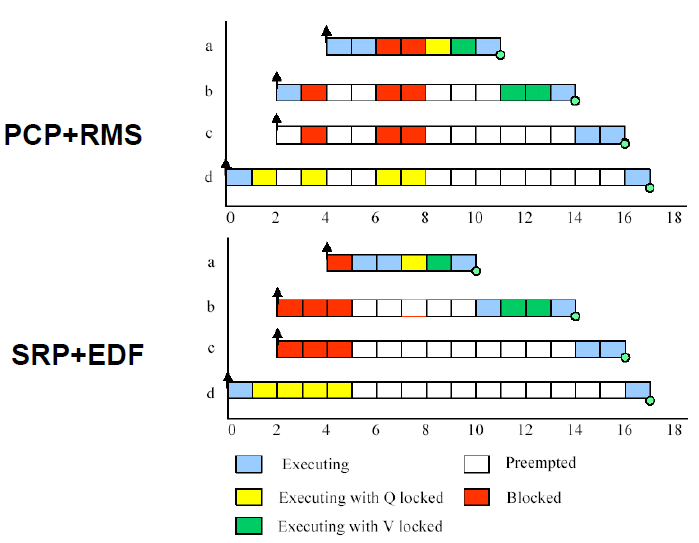
Deadline還原

* + 1. 為實現EDF排程，修改Scheduler尋找最高優先權的部分，以最早截止時間為優先(OS\_SchedNew )，搜尋就緒任務中截止時間最早的任務優先排程，若截止時間相同，則讓周期較長的任務優先執行，會有較好的響應時間及較少的context switch次數，同時須考慮SRP的Ceiling，判斷Preemption Level是否有大於Ceiling。



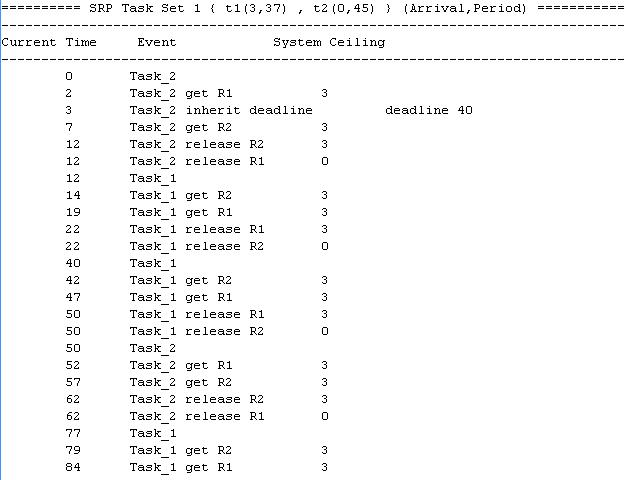
Ceiling判斷

* + **問題與討論**

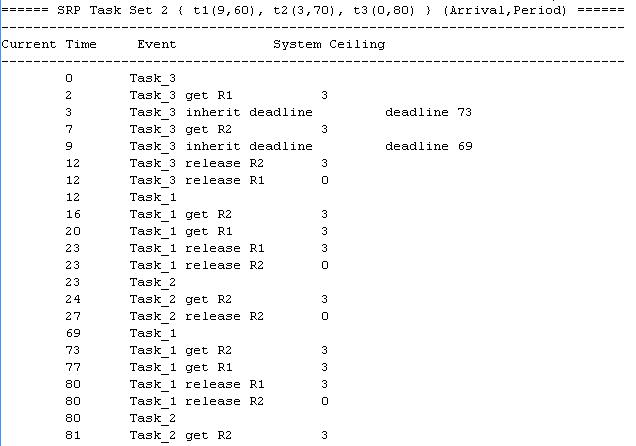


由於PCP與SRP的任務只會被block一次，因當持有資源低優先權任務釋放資源後，高優先權能夠被排程馬上執行而不會被其他低優先全的任務block，所以PCP與SRP的block time 是相同的，唯一不同在於任務檢查權限的時間點，PCP在拿取資源時才會檢查權限，而SRP則是在任務就緒時就檢查，因此任務能夠完整的執行，中途不被中斷，一個任務至多只會有2次的Context Switch，而PCP可能會有2次以上的任務切換而造成額外的Overhead。由上圖的例子中可以看到不管是PCP或SRP其blocking time是一樣的，只是時間點不同，而PCP中可以看出由於是在存取資源時才檢查權限，發現權限不足時又會任務切換至原任務執行等待資源釋放，所以造成了許多不必要的context switch，但是我們可以看出PCP多工任務的平行度是比SRP好的。

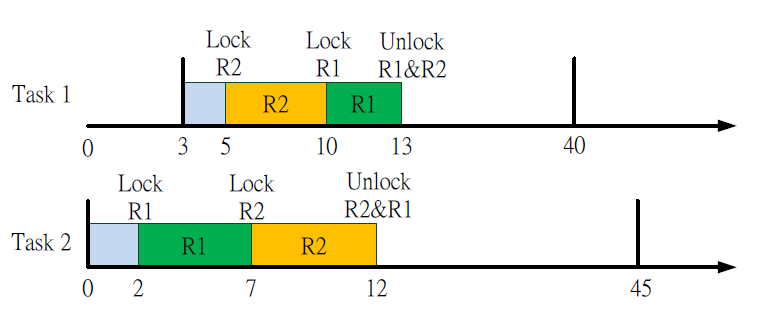
* + **結果(Time Tick 0 - 70)**
    1. Task set 1 模擬:



* + 1. Task set 2 模擬:



* + **Schedulability Analysis**
    1. Task set 1



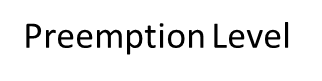
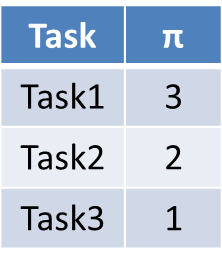
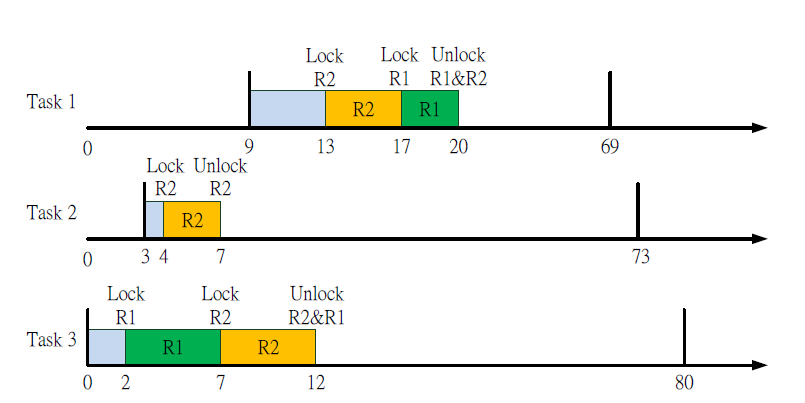


圖1 (a) Input (b) Output

Task set 1 即是典型的Deadlock，task 2先持有了R1，接著task 1抵達並持有了R2，然後task 1又要拿取R1，但R1已被task2所持有，當返回task 2 執行又叫拿取R2，接下來就陷入雙方互等對方釋放資源的情況。使用SRP來解決資源共享的問題時，由上圖可以知道task 2先持有了R1，Ceiling上升到了3 (最高)，當t=3時，task 1抵達，此時發現task 1的preemption level並無高於系統Ceiling，因此無法搶占執行，同時task 2會去繼承task 1的deadline，由於task 1無法搶占資源，所以task 2能夠一次存取所有所需資源到結束，接著會調度高優先權的任務執行，也就不會發生分別持有資源而互等對方釋放資源造成崩貴的情況。

* + 1. Tack set 2



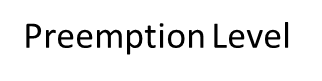
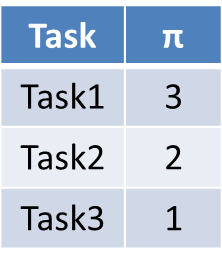


圖2 (a) Input (b) Output

Task set 2 可能造成chain blocking的情況，由最低優先權任務先抵達並持有資源，在釋放資源前，較高優先權的任接續抵達並也要求存取資源，所以導致最高優先權任務抵達要求存取資源時，發現須等較低優先權任務釋放，而中間的任務又等最低優先權任務釋放資源，如此一個等一個，環環相扣，如同鎖鏈一般，造成高優先權任務需要等每個低優先權任務解開才能夠存取。SPR能夠解決chain blocking的問題，如上圖所示，由於task 1來的太慢或是task 2執行時間不夠長所以看不出有chain blocking的狀況，若是task 1緊接在task 2後抵達，且執行時間能夠剛好配合就有可能造成chain blocking情況，而這個例子，剛好不會有這個問題，而是task 1與task 3的deadlock問題。由SPR中Ceiling對資源的權限控管，如同Task set 1的分析，能夠避免掉上述的那些問題。

1. **心得**

本次project主要是實作SRP的部分，練習使用相關工具來達到對資源共享的保護。在程式開發過程中，一定有使用過全域變數，因此就能了解到對與有使用到此變數的程式部分，必須保持該變數的完整及正確性，否則會造成錯誤的運算影響後續的程式部分，甚至會導致系統崩潰，而全域變數就是共享資源的一種例子，因此對於資源的控管是相當重要的。而課堂中介紹了幾種相關的操作函式，為實作出SRP我們須使用Mutex來達到對資源的保護。Mutex本身就有了優先權繼承的概念，因此在這部分不用去修改也少了很多麻煩，只需針對Ceiling及Deadline的部份去做控制，並且在之前改的EDF Scheduler下，加入Ceiling與Preemption Level的比較來控管任務的優先順序。整體來說，有了之前的基礎，這次的Project難度反而是最簡單的，過程中只有遇到了小小問題，系統在運行當中會被Idle任務中斷，後來發現是Scheduler沒加入Ceiling的判斷以及相關的初始。透過程式的撰寫以及實際SRP，對於課堂上的理論有更深刻的理解及內化，也了解到資源共享對一個作業系統來說是非常重要的一件事，沒有好的方案來控管資源，就如同錯誤的政策比貪汙可怕，對後續的發展有嚴重的影響。