競爭與同步

大綱

- 臨界區間與競爭現象
- 鎖定
- 死結
- 資源配置圖與資源配置狀態
- 同步
- 典型的同步問題
- 本章重點回顧

前言

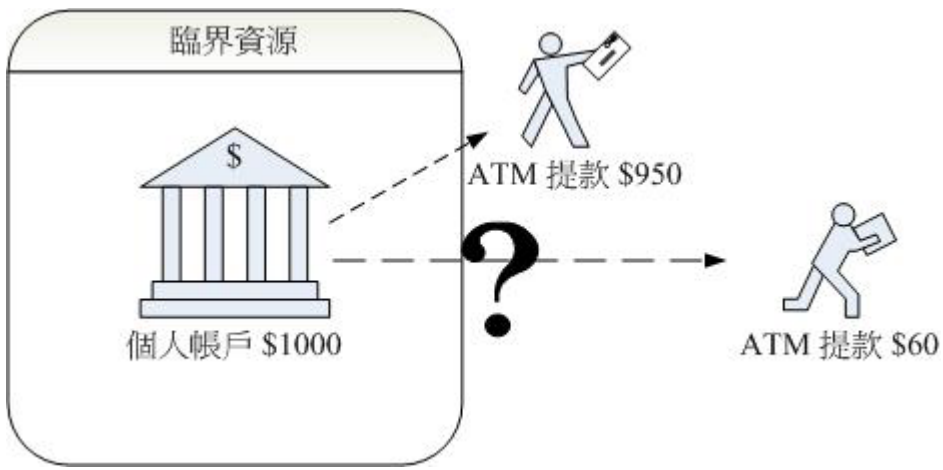
- 現代作業系統的設計概念：
  - 分時多工，以加強資源使用率
    - 資源：例如磁帶機、印表機、CPU、記憶體等
    - 程序可以輪流取得資源的執行權
  - 問題是：資源使用權的『順序』問題
    - 可使用同步機制來保護資源的使用(後面介紹同步)
    - 必須要避免發生死結(Deadlocks)

臨界區間

- 何謂臨界資源(Critical Resource)？
  - 一次只允許一個行程去存取受到保護的資源。而這個受到保護的共用資源稱之為臨界資源 (critical resource)
  - 每個行程用來存取這些資源的程式區段稱之為臨界區間 (critical regions)
- 臨界資源會遭遇到的問題：
  - 競爭現象(Race Conditions)：
    - 多個程序同時要使用這個臨界資源的問題
  - 同步(Synchronization)
    - 避免這種競爭問題的解決方式稱為同步

競爭現象的探討-1

- 有兩個人同時在同一個帳號內提領金額時，會造成什麼問題？
  - 分析：
    - 兩筆記錄都可以執行(因為950, 60都小於1000)
  - 執行過程：(同時進行中)
    - 1000-950=50(剩餘)
    - 1000-60=940(剩餘)
  - 結果
    - 最終剩下940
    - 最終剩下50



競爭現象的探討-2

- i = i+1 值增量的競爭現象
  - 第一個方式可正確執行
  - 第二個方式取得錯誤i值

程序-1	程序-2
取得變數i(5)	
變數i增量 (5 6)	
將i寫回(6)	
	取得變數i(6)
	變數i增量 (6 7)
	將i寫回(7)

程序-1	程序-2
取得變數i(5)	取得變數i(5)
變數i增量 (5 6)	---
---	變數i增量 (5 6)
將i寫回(6)	---
---	將i寫回(6)

原子運算(Atomic Operations)

程序-1	程序-2
取得變數i(5)	取得變數i(5)
變數i增量 (5🕔6)	---
---	變數i增量 (6🕔7)
將i寫回(6)	---
---	將i寫回(7)

- 讓變數的值可以在兩個原子運算中交互運行。
- 鎖定的方式可以達到這樣的目的

鎖定(Locking)

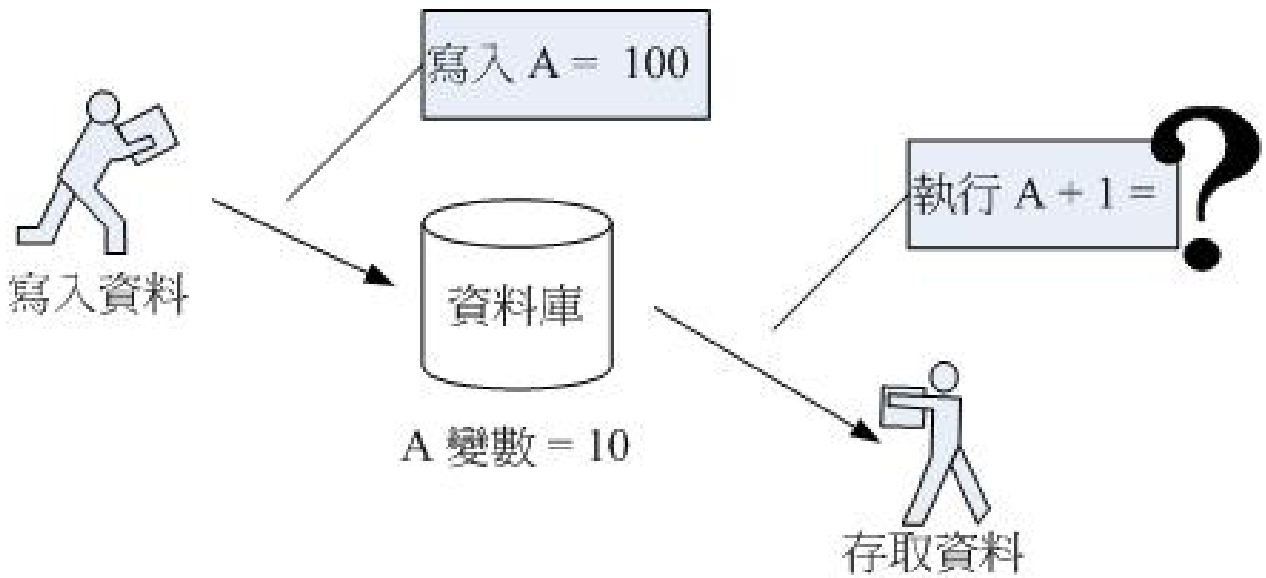
- 多工的處理方式(複習):
  - 設定一個程序佇列池
    - 接受進入佇列池中等待執行的程序
    - 在佇列池中的等待的程序送出執行
- 克服在佇列池中，各程序『同時想要使用同一資源』的方法：
  - 針對資源進行『鎖定』

鎖定

- 鎖定的方式：
  - 當一個程序進行該資源的使用時，
  - 系統『鎖定』該資源，使不被其他程序存取
- 鎖定機制的好處：
  - 鎖定機制的應用可以避免多個請求同時存取共同資源的狀況，也可以免於佇列池中等待的請求產生競賽的現象。

鎖定

- 未加鎖定的情境探討：
  - 資料庫內的A變數同時被更新與讀取時？



鎖定的案例

程序-1	程序-2
嘗試鎖定一個資源	嘗試鎖定一個資源
成功：獲得存取權	失敗：等待中
開始存取	繼續等待
存取結束，釋放鎖定	繼續等待
—	成功：獲得存取權

- 先取得的會有存取權限，
- 未取得的會開始等待，因爲資源被系統鎖定了。
- 請搭配原子運算說明鎖定在何時發生？

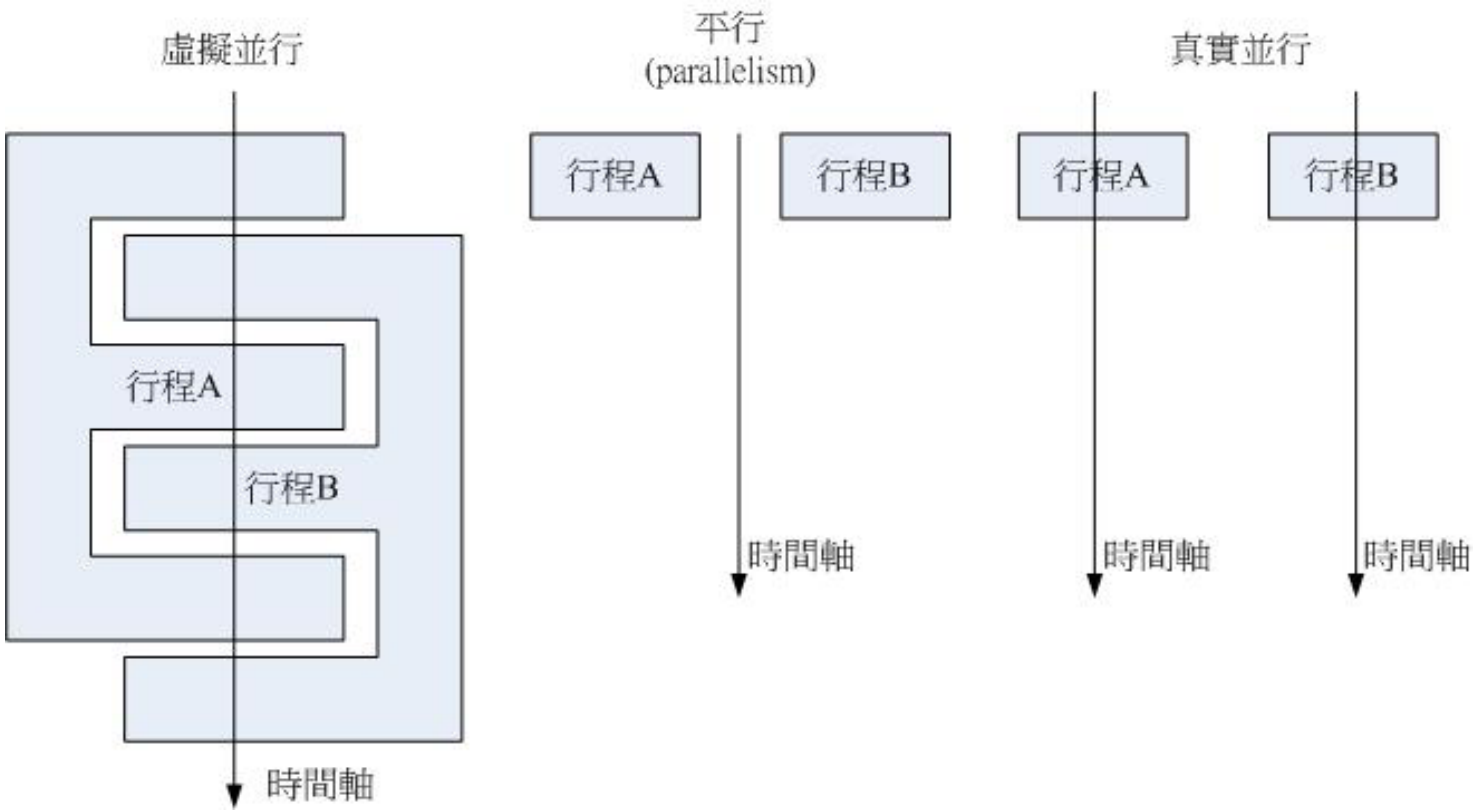
資源存取的並行問題

- 並行的發生→程序同時在運作的情況
  - 程序可能因『先佔式』取得較佳優先權，導致與原本使用該資源的程序『競爭』該資源
  - 單執行緒的處理器上，共享檔案時，可能的『非同步』問題
  - 結論：兩個請求在非同一個時間點，但是彼此卻會在共同的臨界資源中執行者。

不同架構下的並行與平行

- 虛擬並行：
  - 在單處理器環境中，兩個程式『交錯的運作』情況
- 真實並行：
  - 對稱式多重處理器架構下，不同的程序可被不同的處理器運作，有兩個時間軸(因為有兩顆處理器)
- 平行：
  - 單一執行緒讓兩個程序同時間進行，需要程式本身的設計，與並行不相同。

並行與平行間的差異



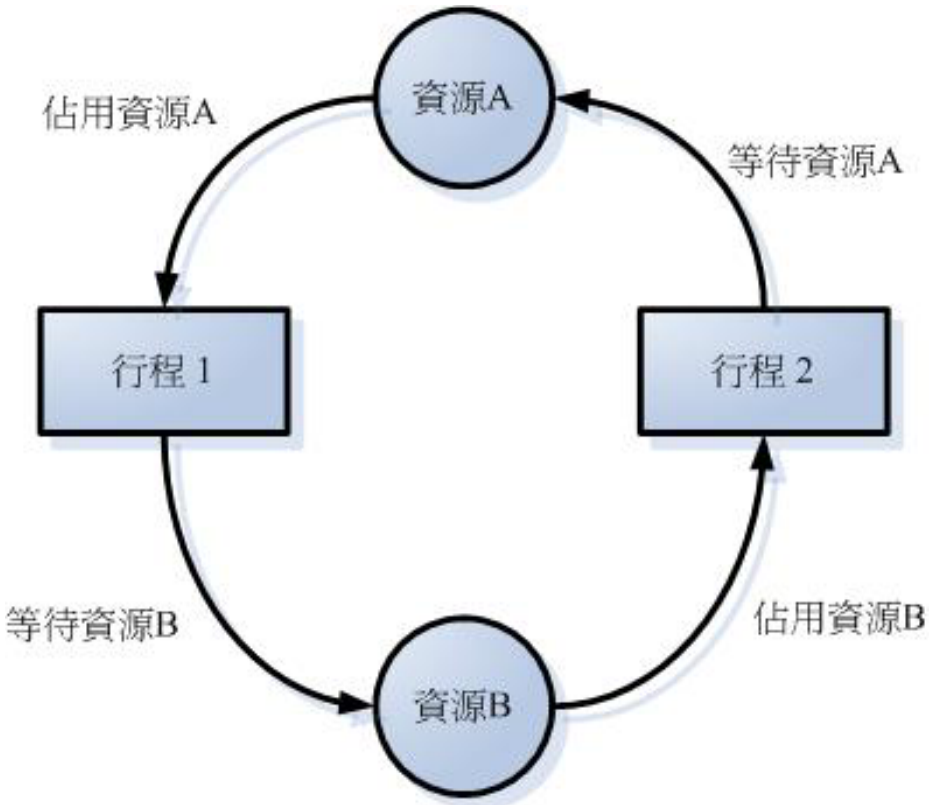
程序交錯運作　　程式本身的設計　　多處理器架構

- Linux作業系統中引發並行的因素：
  - 中斷（Interrupts）
  - Softirqs與Tasklets
  - **核心的先佔式特性**
  - 於用戶層中呈現睡眠等待或是執行同步
  - **對稱多重處理**

死結(Deadlocks)

- 死結：
  - 是指程序間相互等待的一種狀況，並非單一程序的狀態
  - 情境：
    - 程序處於等待獲得某個特定的系統資源，但該資源右臂某一個程序所佔有，則該程序處在『永遠等待』的情況，稱為死結。

死結的情境



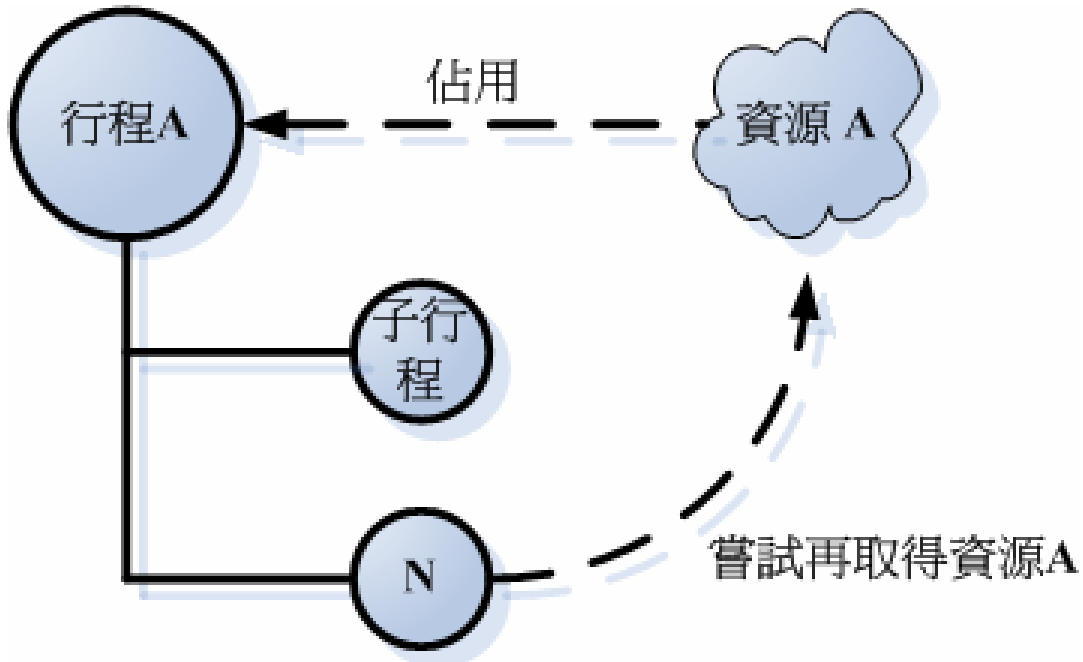
程序2擁有資源B，但需要資源A才能夠運作

程序1擁有資源A，但需要資源B才能夠運作

所以兩個程序永遠在等待對方釋放資源，也就造成死結的狀況

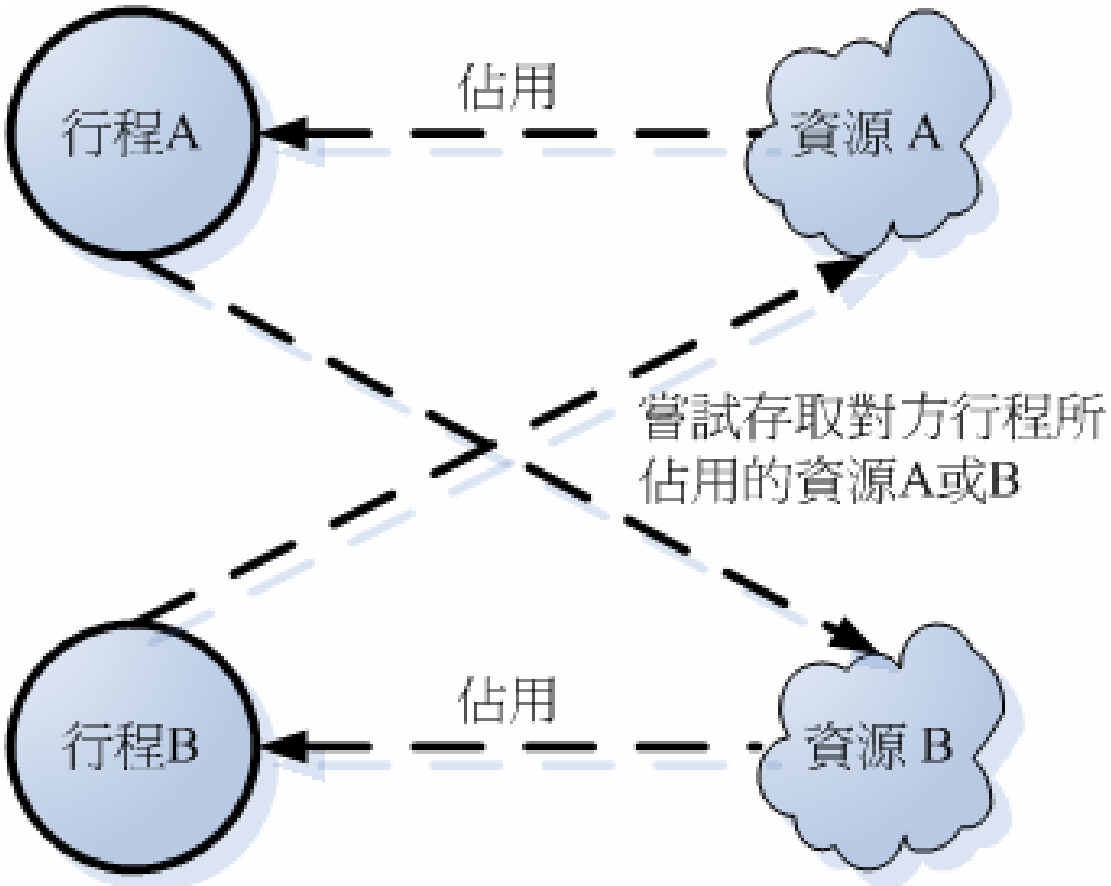
死結的情境—自我死結

- 程序自己已經佔用某資源，卻還要嘗試要求獲得，就造成自我死結

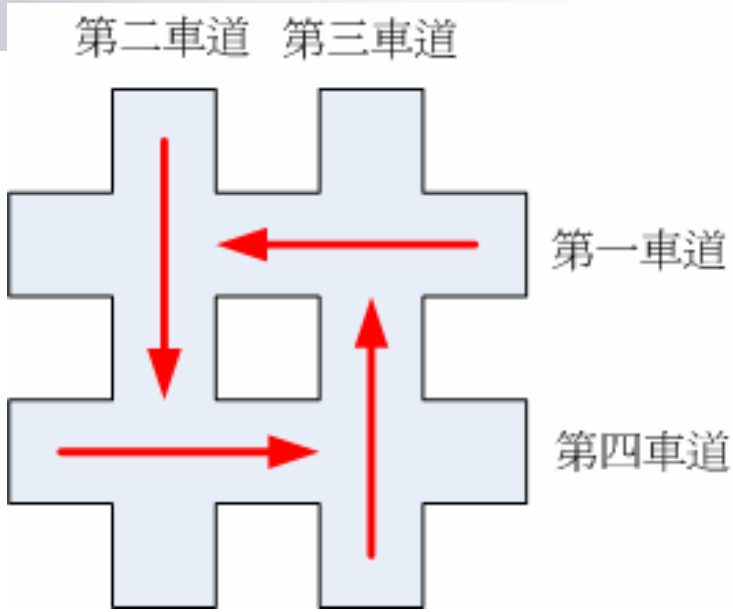


通常是由於子程序的要求所造成的問題

- ABBA死結/致命的擁抱（Deadly Embrace）



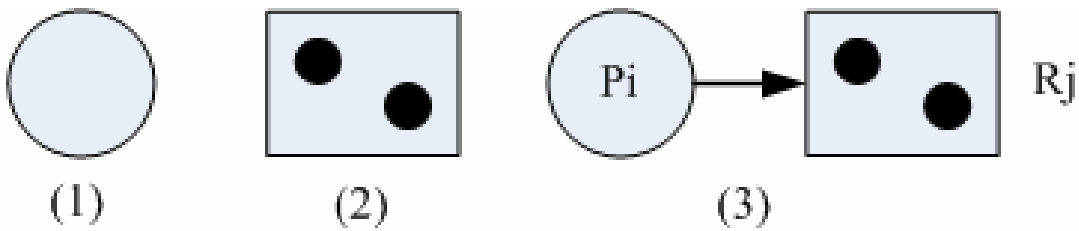
- 死結發生的四大必要條件：
  - 相互排斥（Mutual Exclusion）
    - 程序所共用的資源中，至少有一個是不能被兩個以上程序共同存取的情況
  - 佔用與等待（Hold and Wait）
    - 當程序已經獲得某項資源，可還需要其他系統資源才能夠繼續進行，但該資源被佔有，而進入等待時
  - 不可先佔（No Preemption）
    - 程序所佔用的資源無法被其他先佔式程序所搶走時
  - 循環等待（Circular Wait）
    - 程序之間互相等待對方的情況(A→B→C→A)



- 車道現象：
  - 相互排斥
    - 每個路口只允許單向
  - 佔用與等待
    - 路口處，車輛各自佔用一個車道，並等待另一車道
  - 不可先佔
    - 均需等另一車道通過後才能通過，無法搶先
  - 循環等待
    - 四車道互相等待對方四→一→二→三→四→...

資源配置圖與資源配置狀態

- 資源配置圖，主要是用來處理避免死結的產生，但這類的圖形比較適合每個資源都是只有一個的情況下。

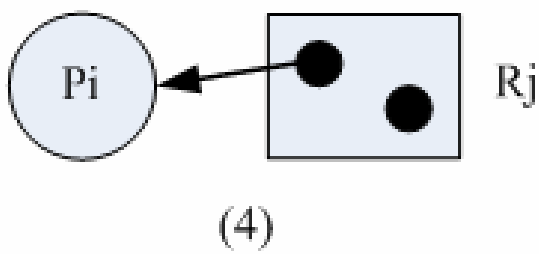


(1)圓形：代表程序

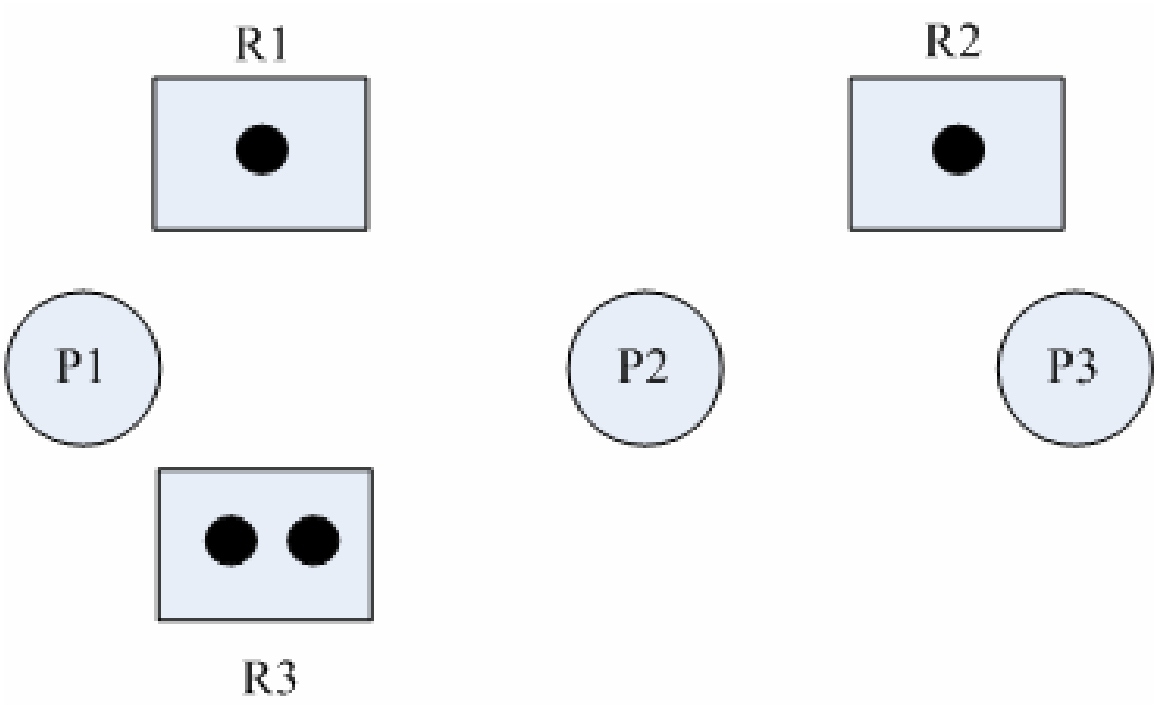
(2)方形：每種資源的類型，裡面的圓點則是該資源的數量

(3)要求模式：Pi要求j

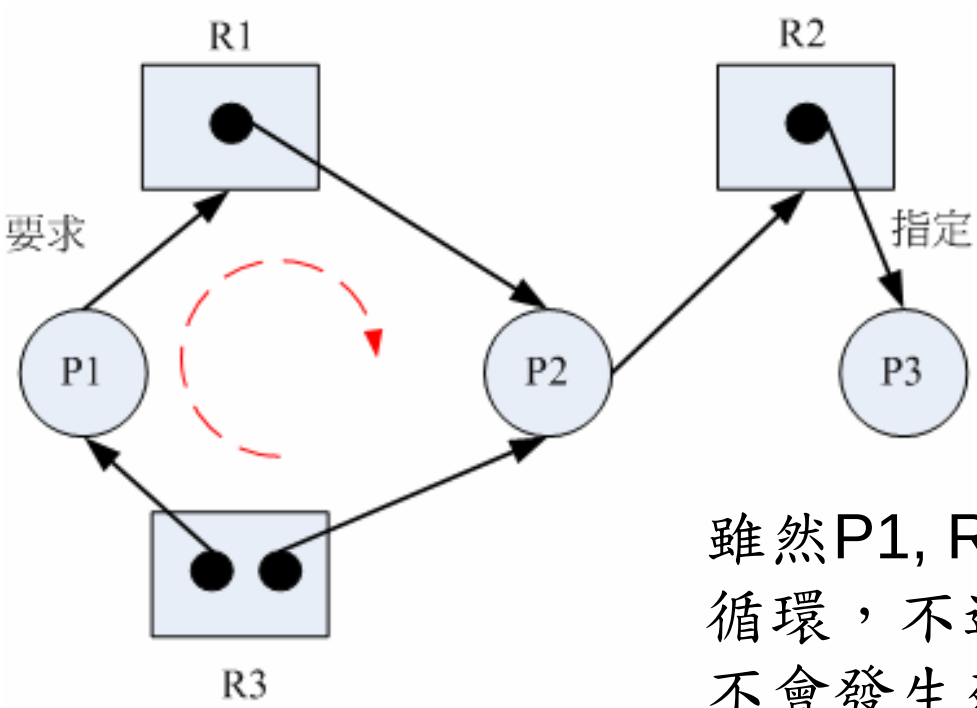
(4)資源指定：Rj分配給Pi



- 資源配置圖初始狀態



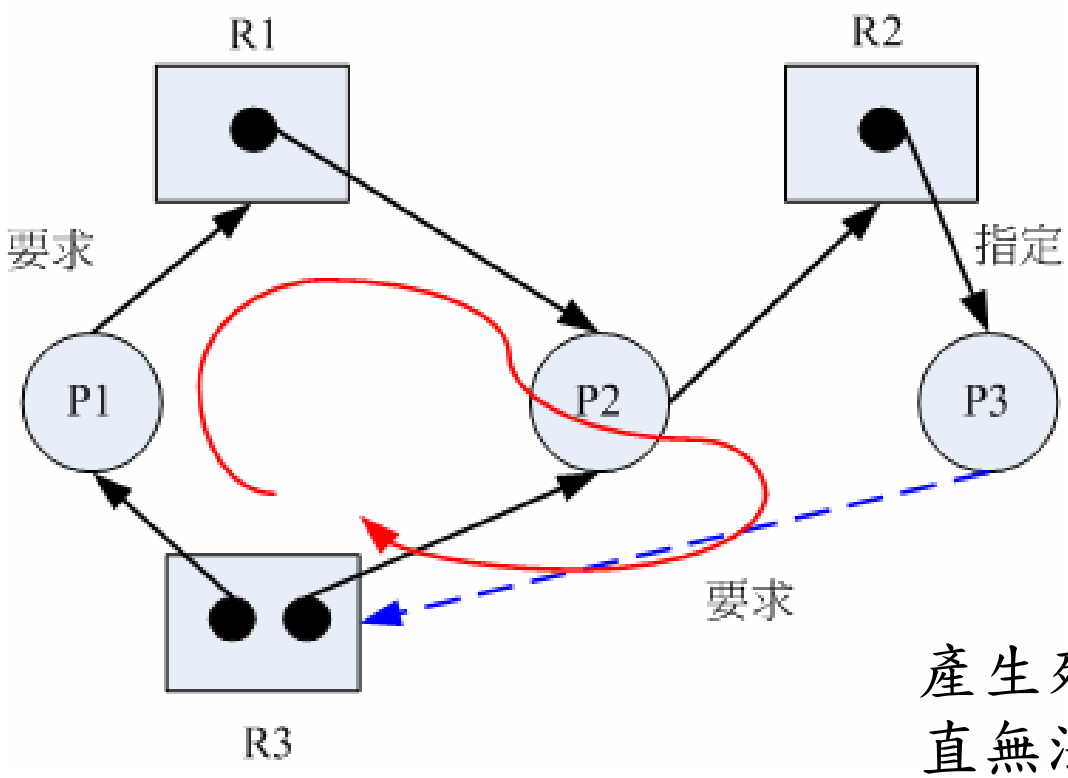
資料配置圖範例一



雖然P1, R1, P2, R3之間有循環，不過R3有兩個，所以不會發生死結。因為

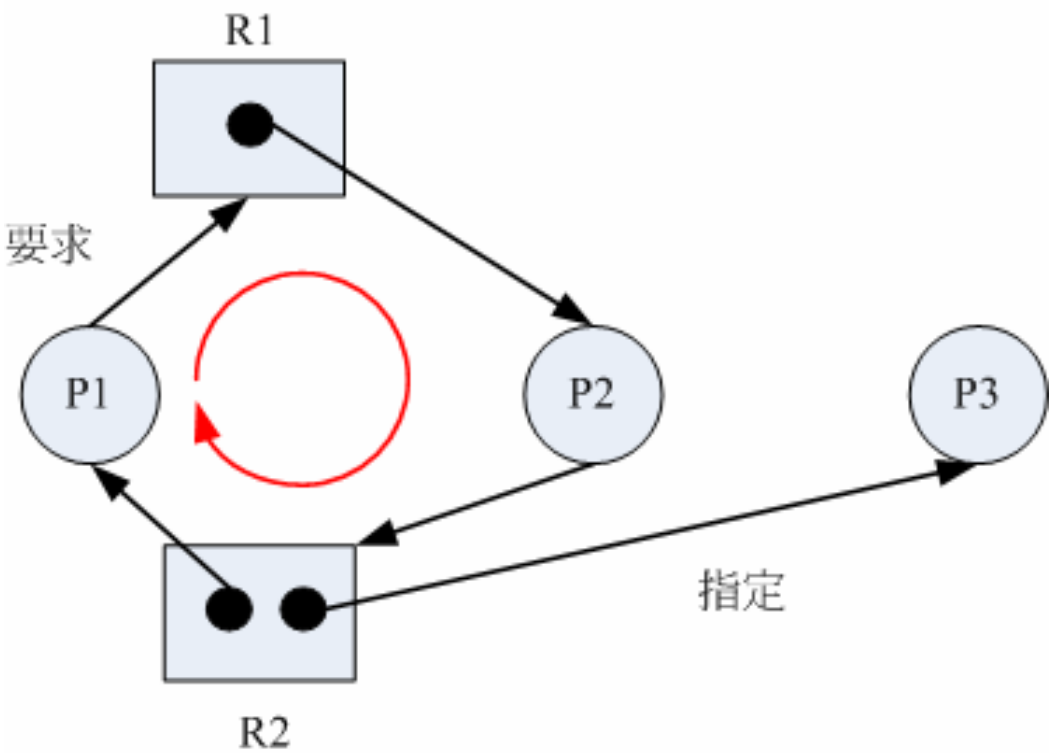
P3可執行完，然後P2可取得R2繼續執行完，最後P1可執行完畢。

資料配置圖範例二



產生死結了，因為P3一直無法取得R3，導致P3, P2, P1互相等待了

資料配置圖範例三



雖然開始P1, P2無法進行，所以造成循環。

但因為P3可自行運作，等待P3做完後，P2可取得R2的使用權，進一步完成後，P1就能夠取得R1來運作完成了。

所以沒有死結。

同步

- 為何需要同步？
  - 程序間對臨界資源的使用可能有競爭效應
  - 為了效率，程序間會有並行的機制
  - 但是並行可能因為資訊交換不正確，導致錯誤。因此可利用同步來處理這個問題。
    - (所以，所謂的『同步』意指讓記憶體內的某些變數在不同的程序間能夠同步)

同步的機制

- 同步的機制可分：
  - 硬體機制
    - 測試與設定：Test-and-set
    - 比較與轉換：Compare-and-swap
  - 號誌機制(Semaphore)
    - 二元號誌(Binary Semaphore)
    - 計數號誌(counting semaphore)

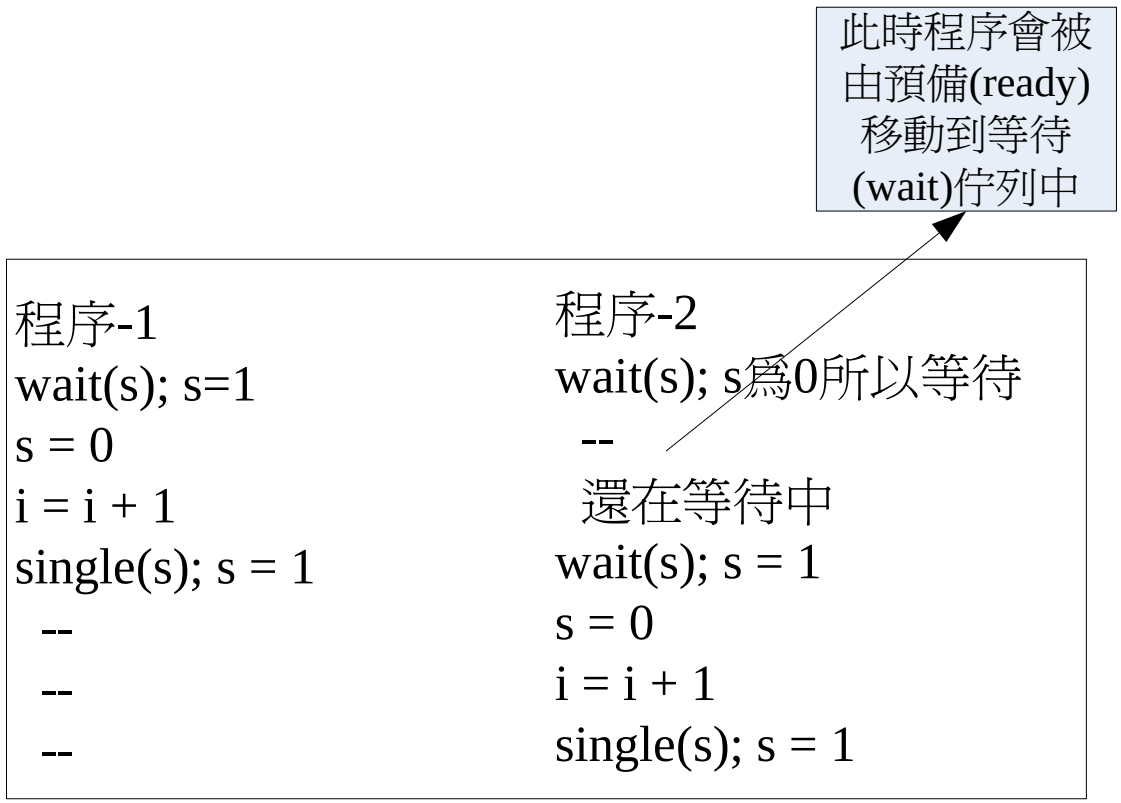
硬體機制(Test-and-set)

- 程序要使用臨界區間時，會先test-and-set檢查lock布林值『boolean lock=false;』

程序-1	程序-2
lock = false	lock = true
取得變數i的值(5)	忙碌等待
lock = true	--
針對變數i增量(5->6)	--
將i寫回(6)	lock = false
	取得變數i的值(6)
	lock = true
	針對變數i增量(6->7)
	將i寫回(7)

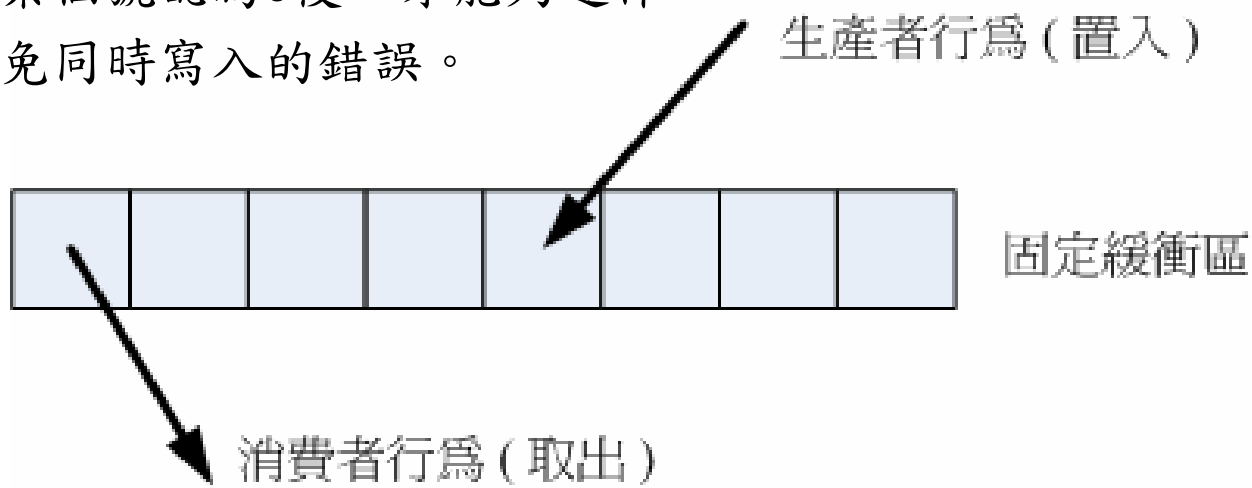
二元號誌的機制

- 透過wait()與signal()這兩個必須一次完成的佇列來處理

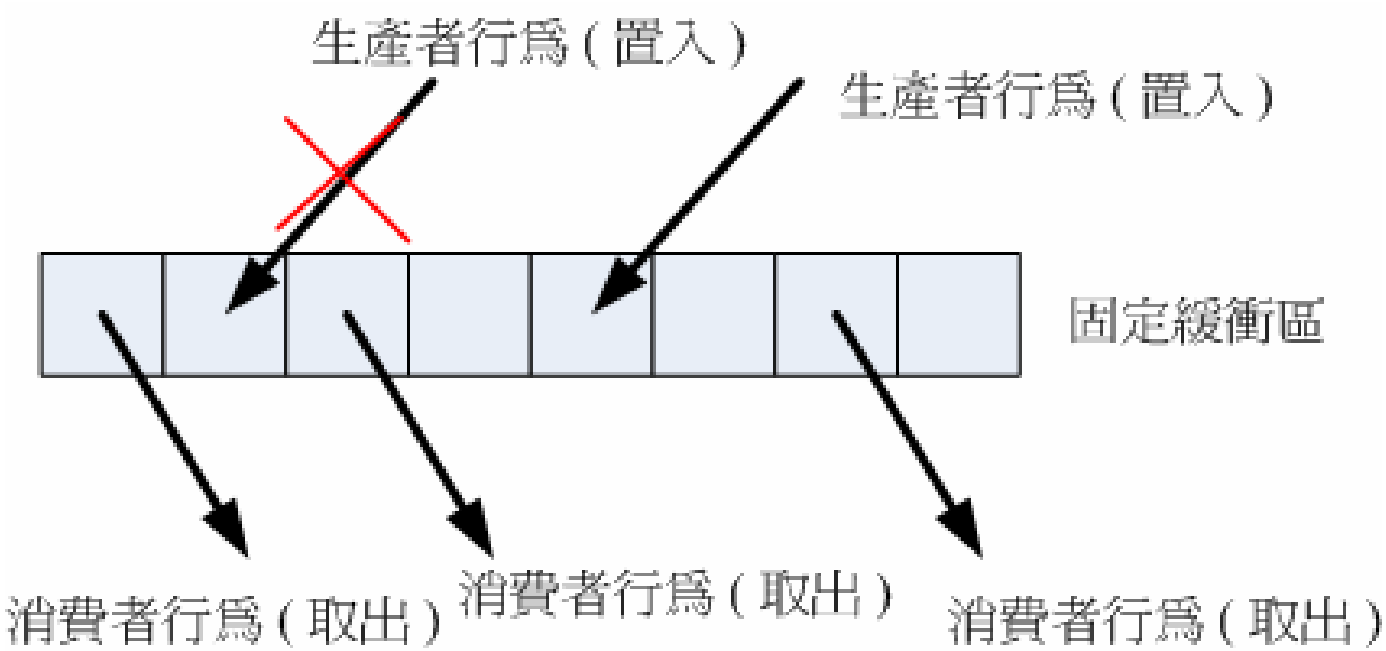


利用同步機制解決問題

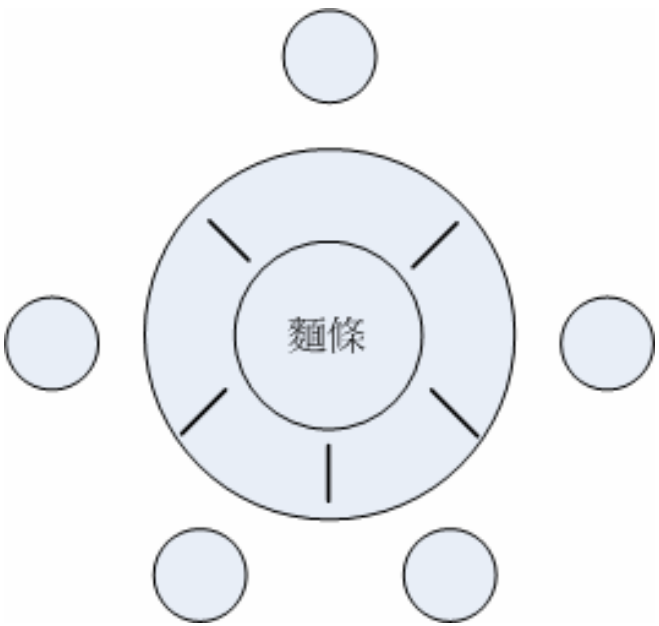
- 有限緩衝區問題（Bounded-Buffer Problem）
  - 利用二元號誌來處理
    - 要進行置入或取出時，
    - 必須要讓某個號誌為0後，才能夠運作，
    - 就能夠避免同時寫入的錯誤。



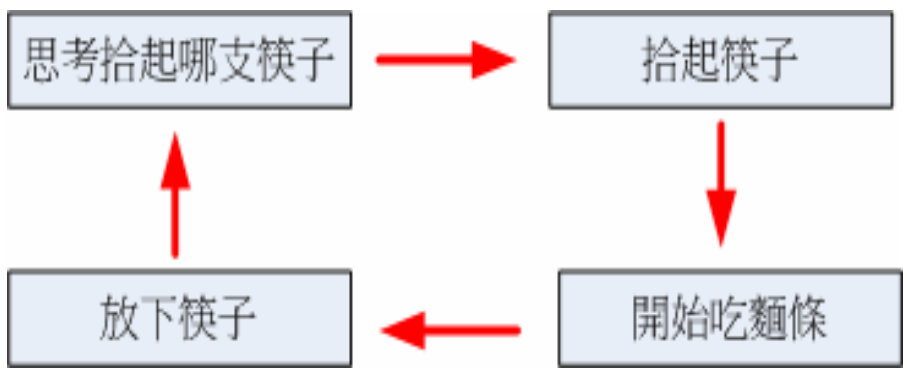
- 讀取者與寫入者問題（Readers-Writers Problem)
  - 與前一個有限緩衝區類似，只是前一個只有一組生產/消費，這裡則是有多個生產/消費行為。
  - 透過類似的行為來處理，只是變數就會變的比較多。



- 哲學家進餐的問題（Dining-Philosophers Problem）
  - 一個圓桌只有五根筷子，有五個哲學家
  - 哲學家只會思考與吃麵
  - 若哲學家飢餓時，會拿最接近的兩根筷子吃麵，且一次只能拿一根(非左即右)
  - 吃完後會放回原處



筷子是臨界資源
哲學家是程序，可能會競爭筷子
筷子需要用二元號誌來『鎖定』
哲學家會在兩側筷子沒有被鎖定下，才會開始進行取筷子的動作



本章重點回顧

- 了解何謂臨界區間與競爭現象。
- 了解鎖定的機制與透過鎖定機制去解決競爭的現象。
- 了解到並行與平行的執行差異。
- 了解到何謂死結，與死結的發生跟類型。
- 了解到同步問題的處理方式。