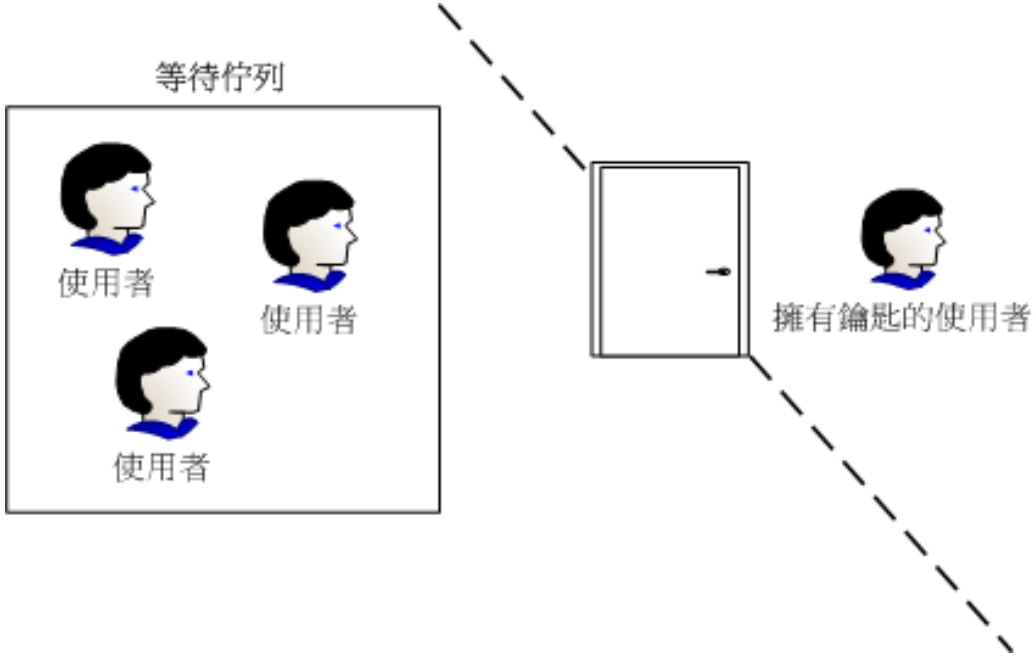
競爭與同步解決方案

大綱

- Linux核心提供的號誌應用
- 死結的預防與處理
- 死結的避免與銀行家演算法
- 忽略死結與死結狀態恢復
- 核心的鎖定與先佔式的失效
- 本章重點回顧

Linux核心提供的號誌應用

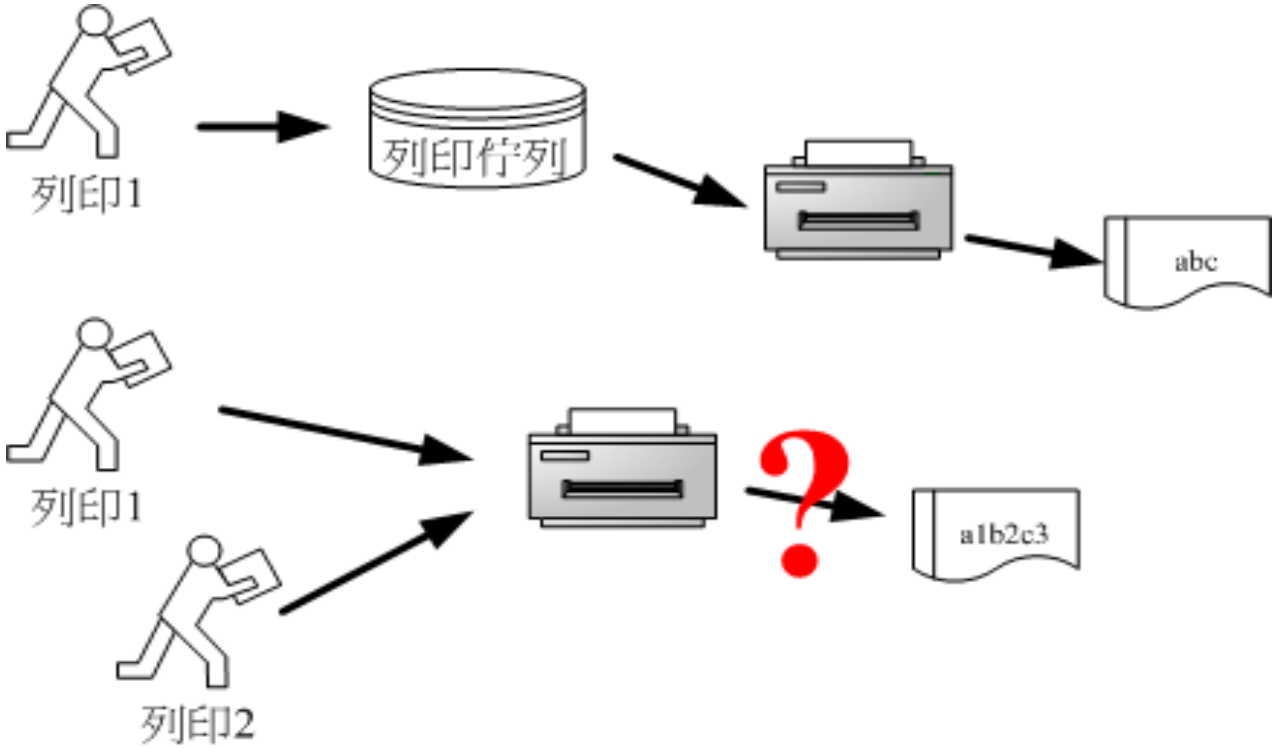
- 號誌的運用 - 門與鑰匙範例



死結的預防與處理

- 作業系統在處理死結的四種方式：
  - 預防（Prevention）
  - 避免（Avoidance）
  - 偵測與回覆（Detection and Recovery）
  - 完全不理

- 預防死結的方法就是確保作業系統於運行期間中，四大條件並不會同時成立。
  - 相互排斥

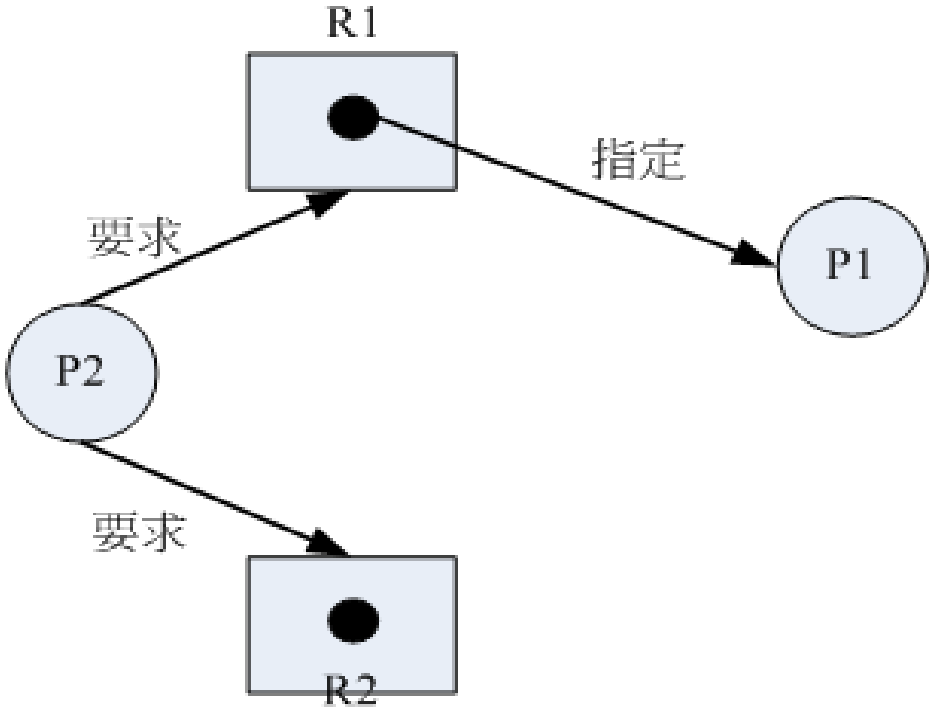


- 佔用與等待
  - 要求行程執行前，須先對全部的資源進行請求並獲得分配。
  - 在請求額外的資源前，須先釋放原本持有的資源。
- 不可先佔
- 循環等待

死結的避免與銀行家演算法

- 死結的避免（Deadlock Avoidance），就是作業系統本身可以依據現有可使用的系統資源，與現有已被行程擁有的資源狀況，以及未來其他行程的要求，或是釋放佔有的資源種種因素的考量下，來決定是否同意目前提出資源存取要求的行程，是否可以擁有資源。

- 安全狀態的資源分配順序
  - 當作業系統呈現在安全狀態中時，系統是絕對不會產生死結。
  - 當作業系統呈現在不安全狀態中時，系統不一定會產生死結。
  - 死結的產生一定是由不安全的狀態中所衍生的。
  - 只要使作業系統呈現在安全狀態中，系統便可以避免死結的產生。



- 銀行家演算法（Banker’s Algorithm）
- 銀行家演算法只能說是一種「理想」的執行方式，在作業系統中並非是一個很實用的演算法方式
- 銀行家演算法中使用兩個向量與三個矩陣資料結構
  - 代表可用資源數量的向量A（Available）

    $A = [3,3,2]$
  - 代表行程是否可以結束的向量F（Finish）

$$F = \begin{bmatrix} false, false, false \\ true, true, true \\ false, true, true \\ true, true, true \\ false, true, true \end{bmatrix}$$

- 代表行程所需最大資源數量的矩陣M（Max）

$$M = \begin{bmatrix} 7 & 5 & 3 \\ 3 & 2 & 2 \\ 9 & 0 & 2 \\ 2 & 2 & 2 \\ 4 & 3 & 3 \end{bmatrix}$$

- 代表行程目前所佔用的資源數量矩陣H（Hold）

$$H = \begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

- 代表行程目前尚需求多少資源數量矩陣N（Need）

$$N = \begin{bmatrix} 7 & , 4 & , 3 \\ 1 & , 2 & , 2 \\ 6 & , 0 & , 0 \\ 0 & , 1 & , 1 \\ 4 & , 3 & , 1 \end{bmatrix}$$

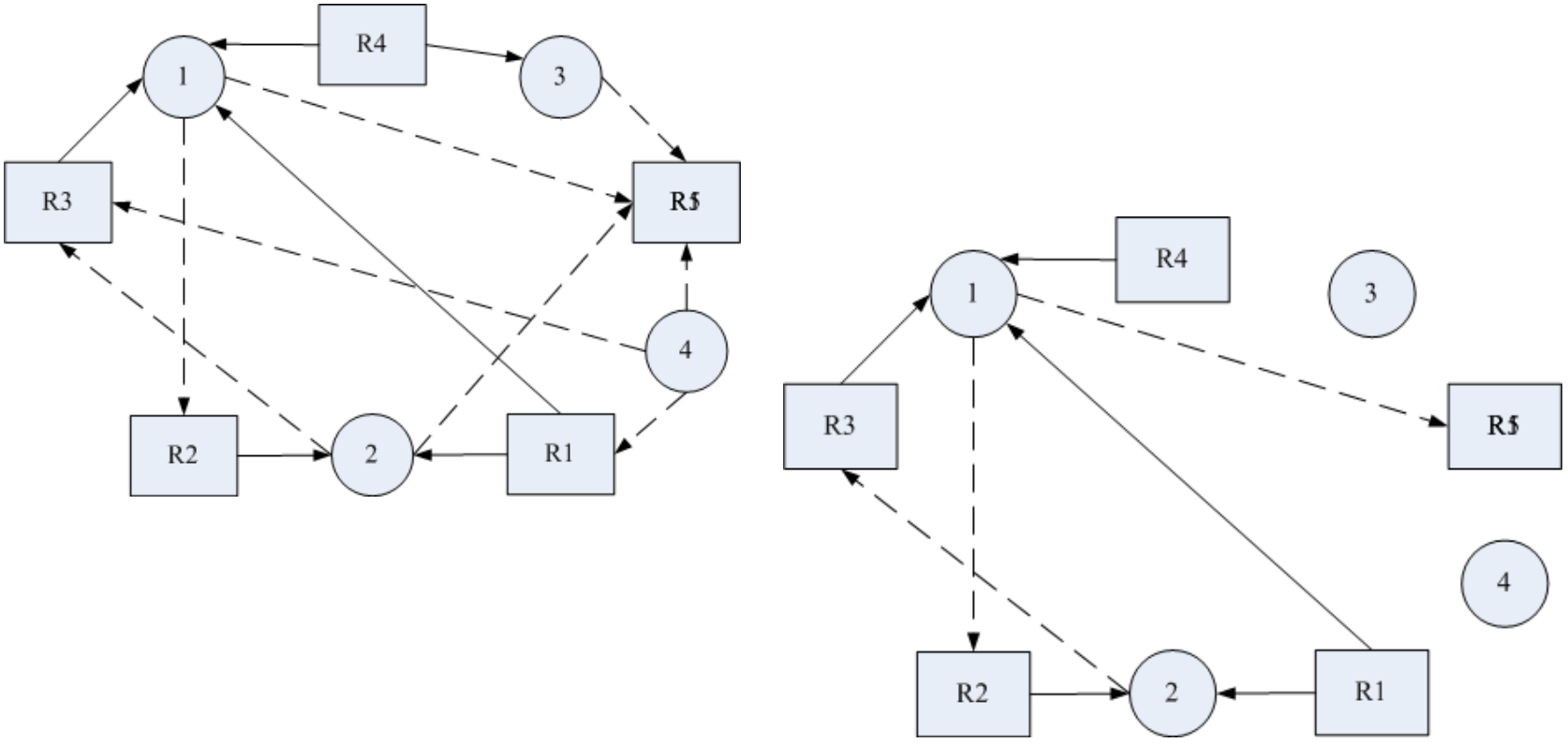
- 如果這時作業系統上的行程，對資源的需求值有所改變時，也可以透過銀行家演算法來定義一個向量R來代表改變的資源要求，其中向量R代表行程所要求的資源種類與個數。

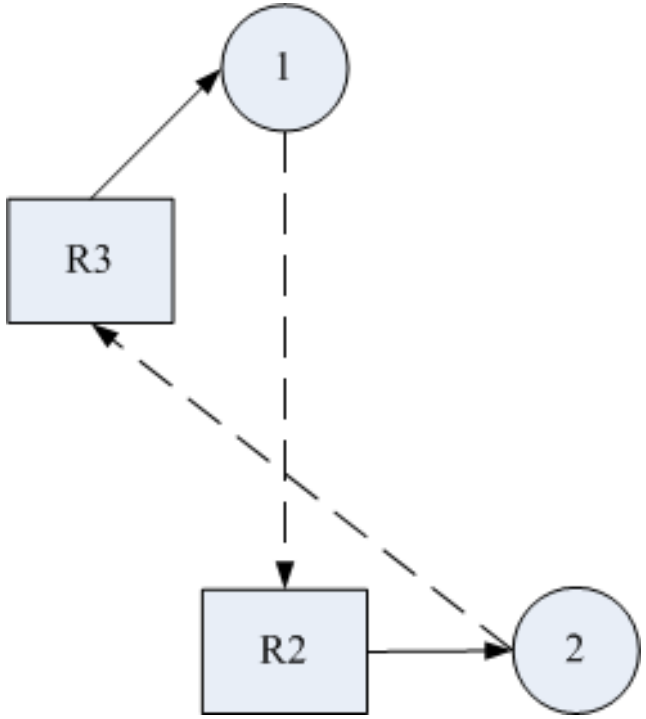
$$R_{P(1)} = [1,0,2]$$

  - 新的向量A=向量A-向量R
  - 新的矩陣H的第n列=矩陣A的第n列+向量R
  - 新的矩陣N的第n列=矩陣N的第n列-向量R

忽略死結與死結狀態恢復

- 偵測死結的模式處理這個死結現象





- 結束造成死結的行程 - 造成作業系統內部狀態的不一致性（Inconsistent）
- 強制行程釋放佔有的系統資源 -飢餓（Starvation）

核心的鎖定與先佔式的失效

- 沒有一個適當的鎖定機制運行著的影響，便會在系統中產生競爭現象（Race Conditions）
- Linux核心中最普遍的鎖定機制 - spinlock
- 除了spinlock之外，Linux作業系統還提供一個透過kernel_flag所展現的核心鎖定機制

本章重點回顧

- 了解Linux作業系統核心中所提供的號誌運用方式。
- 了解到死結的預防與處理方式。
- 了解到死結的避免與銀行家演算法的運行方式。
- 了解到Linux作業系統提供的鎖定機制與先佔式功能的失效所造成的影響。