第3章 行程間通訊與同步

合作行程

- 會與其他行程共享資訊，或是影響其他行程所執行之指令的行程
- 行程間通訊：合作行程之間必定需要某些溝通的機制，以傳遞所需的指令或資料
- 同步：行程合作過程中所牽涉到的協調問題與處理機制

3-1 行程間通訊（IPC）

- 共用的檔案：行程間最常見的資訊共享方式，傳遞的資訊量僅受限於檔案系統的最大長度
- 共享記憶體：由作業系統提供某種機制，讓不同行程可以同時存取到某塊記憶體
  - 比共用檔案的傳遞速度快，但長度比較受到限制
  - 執行緒本身就內含了共享記憶體的概念
- 信號：當行程A想要告知行程B某事發生或完成時，可以傳送特定的信號給行程B
  - 可以用來進行事件的告知，但是如果需要傳送特定資訊時，還需要搭配其他的行程間通訊機制

訊息傳遞系統-直接溝通

- 溝通雙方必須要知道對方的身分，
- 一個通訊鏈結只有2個行程參與，可單向/雙向
- 系統通常會提供2個函式
  - send_process(pid, message)
  - receive_process(pid, message)

訊息傳遞系統-間接溝通

- 通訊鏈結可以由數個行程共享
- 涉及許多權限管理的問題
- 系統通常會提供4個函式
  - create_mailbox(mbx)
  - delete_mailbox(mbx)
  - send_mailbox(mbx, message)
  - receive_mailbox(mbx, message)

阻隔式傳送 vs. 非阻隔式接收

- 不論是直接或間接形式的溝通，都會面臨收送雙方不同步的情況。
- 阻隔式傳送/阻隔式接收：讓等待的一方進入懸置狀態，直到資訊被接收後再繼續
- 非阻隔式通訊：
  - 非阻隔式傳送：傳送端行程送出訊息之後，不必等訊息被接收，就可以繼續執行後面的動作
  - 非阻隔式接收：接收端行程在嘗試讀取訊息之後，不論是否有收到訊息，都可以繼續執行後續的動作
- 當傳送端與接收端都是使用阻隔式通訊時，兩者之間就會同步

行程間通訊的設計考量

- 通訊鏈結的類型：直接或間接
- 共享通訊鏈結的行程數目
- 通訊鏈結的方向性：單向或雙向
- 訊息長度：固定長度或可變長度
- 傳送的是訊息複本或是訊息的參照位址
- 單方行程片面終止時要如何處理

3-2 同步

- 合作行程之間可能在不同的時機點發生交互影響
  - 在單處理器系統：合作行程是根據排程的結果輪流在CPU中交錯執行，使得結果看起來好像是同時執行一樣
  - 在多處理器系統：合作行程不但可能交錯執行，甚至可能確實是同時執行
- 同步機制：確保在多個合作行程同時執行的環境中，不同行程的運算不會相互干擾

合作行程間的可能問題

- 飢餓
- 死結
- 競賽狀況
  - 共享資料的不一致現象

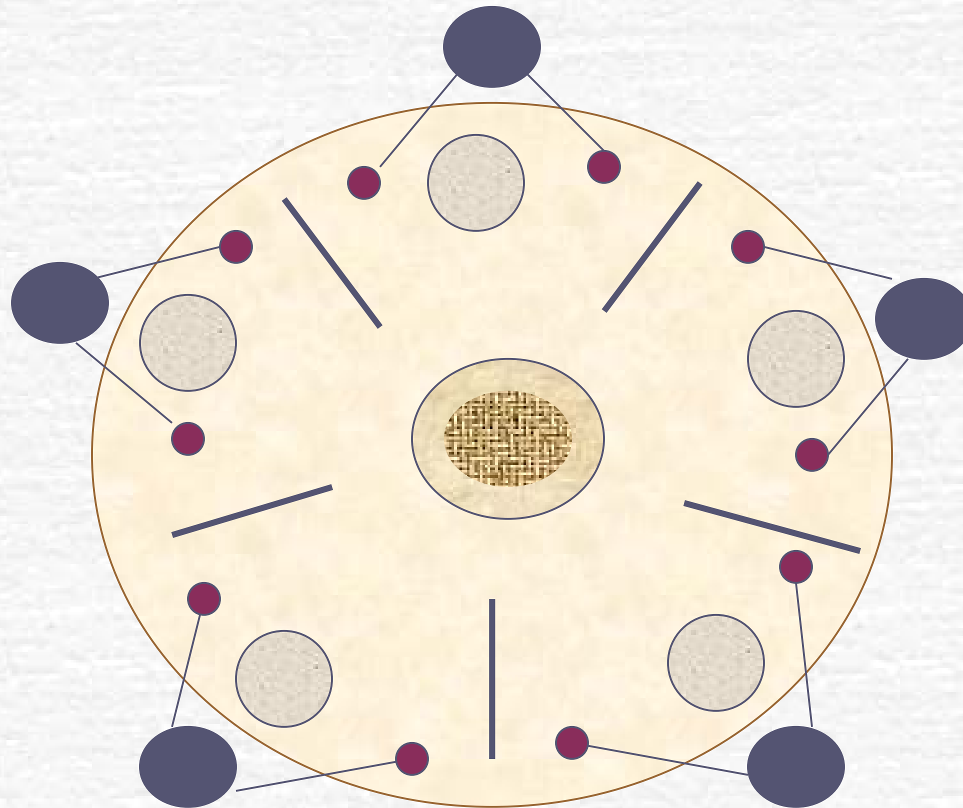
3-4 同步問題的經典範例

- 哲學家的晚餐問題

哲學家的晚餐問題

- 假設有5個哲學家圍坐在圓桌上共進晚餐，桌上只有5支筷子，每2個哲學家共用1支筷子
- 哲學家們坐在餐桌上仍舊思考著哲學問題
- 當哲學家覺得餓的時候，他會從左右各拿起一支筷子來進食
- 拿齊2支筷子的哲學家們可以同時進食，但是不能搶奪別人手中的筷子
- 吃飽之後，哲學家會將2支筷子都放下，繼續思考哲學問題，其他的哲學家則可以使用放下的筷子

圖3-9 哲學家的晚餐



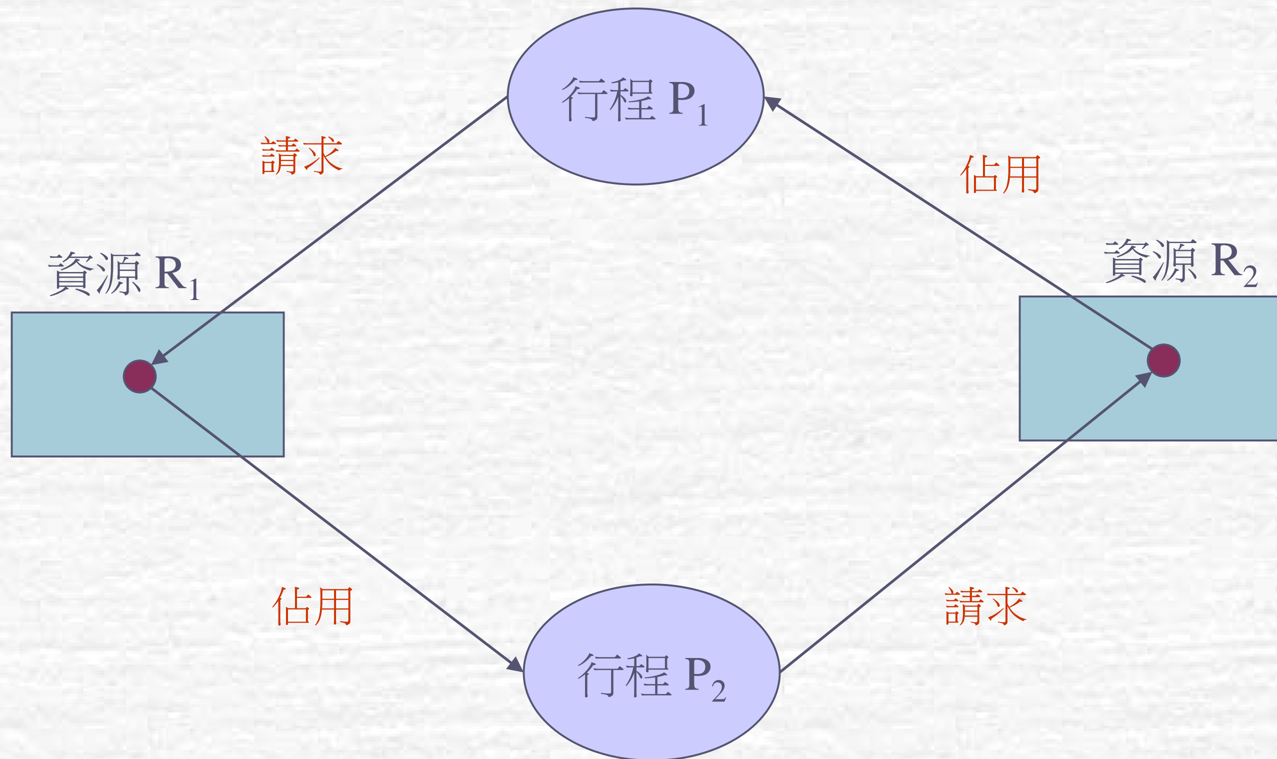
哲學家的死結問題

- 當每個哲學家都拿起右手的筷子，並且等待左手的筷子時，就會發生死結
- 可能的解決方法：
  - 在桌上放置n+1支筷子，或是限制最多只有n-1位哲學家可以同時進餐。
  - 規定哲學家必須要同時拿起左右2支筷子。
  - 規定奇數座位的哲學家要先拿左邊的筷子，而偶數座位的哲學家要先拿右邊的筷子。
- 這是在應用程式層次，針對問題特性來解決死結問題；另一種是在系統層次來解決

3-6 死結

- 只要雙方都掌握了某些資源，也都需要對方手上的另外一些資源，就可能發生無限等待，而造成死結問題
- 死結發生的條件：
  - 互斥：資源必須是不可共享的
  - 佔用且等候：行程在等候其他資源的時候，仍舊佔用某些資源
  - 不可搶先：不能強制從已經取得資源的行程手中搶走它的資源
  - 循環等待：兩個或更多行程形成一個封閉迴路，每個行程都在等待迴路中的下個行程所佔用的某些資源

圖3-10 循環等待的資源配置圖



死結的處理方式

- 預防
- 避免
- 偵測

預防死結

- 預防互斥：不可行；無法同時共用的資源本身一定具有互斥的特性
- 預防佔用且等候：要確保每個行程只有在不佔用 資源的情況下，才可以請求資源
- 預防不可搶先：僅適用於可以輕易地保存狀態的資源
- 預防循環等待：為資源定義一個線性的優先權順序，規範行程在請求資源時，必須依照優先順序提出請求

避免死結

- 如果判斷某項資源配置請求會導致死結發生，就不要進行這項配置
- 安全狀態：系統能夠以某種安全執行序列為每個行程配置資源，並且避免死結的狀態
- 安全執行序列：必須保證「在序列後方之行程所需的資源數量，一定會小於系統目前尚未配置的資源數目，加上前方所有行程所持有的資源總數」
  - 當系統中存在這樣的一個安全序列< P1, P2,…Pn>時，我們可以確定當行程P1到Pi（1 <= i < n）結束的時候，行程Pi+1一定可以取得所有需要的資源
- 當系統中找不出來這樣一組序列時，表示系統處於不安全狀態，也就是有發生死結的**可能**

圖3-11 安全的資源配置

行程	需要R的總數	持有R的數目	目前還需要R的數目
P_1	10	5	5
P_2	3	2	1
P_3	8	2	6

系統中擁有的R總數=12，目前閒置的R總數=3

安全執行序列

行程	配置後閒置的R總數	執行完畢後，閒置的R總數
P_2	2	5
P_1	0	10
P_3	4	12

圖3-12 不安全的資源配置

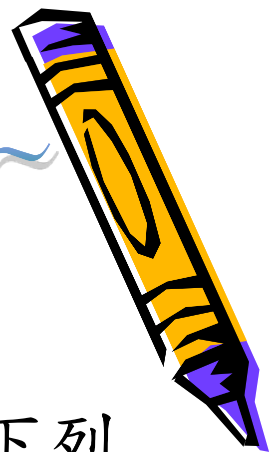
行程	需要R的總數	持有R的數目	目前還需要R的數目
P_1	10	5	5
P_2	3	2	1
P_3	8	**3**	**5**

系統中擁有的R總數=12，目前閒置的R總數=2

可能的配置情況：

行程	配置後閒置的R總數	執行完畢後，閒置的R總數
P_2	1	4

因為 4 小於P_1 、P_3所需的數量 5，所以陷入死結狀態。



課堂練習

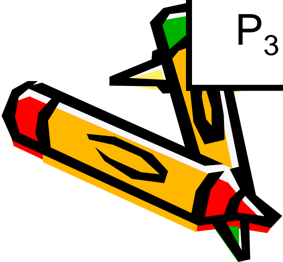
- 請根據行程的資源需求與配置情況，判斷下列系統是否處於安全狀態；如果是的話，請寫出它的安全執行序列

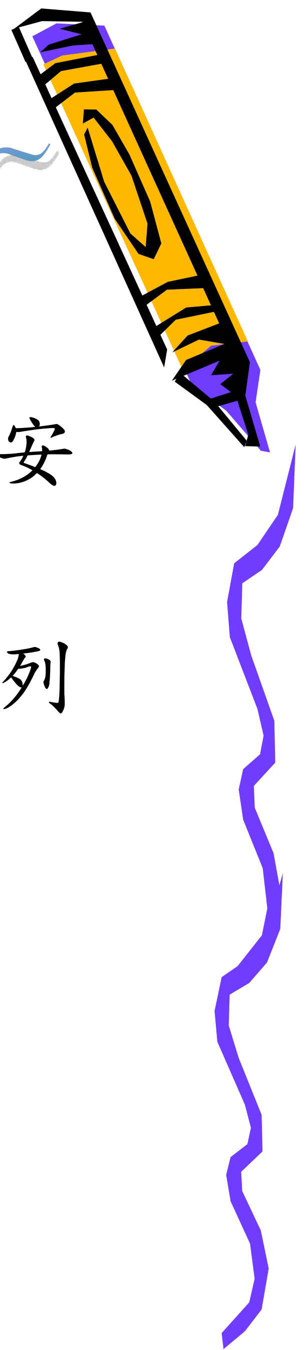
系統一：資源R的總數為11

行程	需要R的總數	持有R的數目	目前還需要R的數目
P_1	10	6	4
P_2	3	1	2
P_3	8	2	6

系統二：資源R的總數為11

行程	需要R的總數	持有R的數目	目前還需要R的數目
P_1	10	1	9
P_2	8	7	1
P_3	7	2	6





練習解答

- 系統一：處於不安全的狀態，找不到安全執行序列
- 系統二：處於安全狀態，安全執行序列為$P_2 \rightarrow P_3 \rightarrow P_1$

銀行家演算法

- 利用安全狀態的概念，藉由防止系統進入不安全狀態，來避免死結問題
- 主要包含兩個部份
  - 安全演算法
  - 資源請求演算法

圖3-13 銀行家演算法範例

行程	需要R的總數 Max			持有R的數目 Allocation			系統尚未配置的R總數 Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	8	0	2	1	0	0	3	1	2
P_2	5	2	3	3	0	0			
P_3	1	2	4	0	1	2			
P_4	7	7	7	2	5	2			
P_5	3	3	5	0	2	3			

表格現況➔安全狀態，存在安全序列： $P_5 \rightarrow P_3 \rightarrow P_2 \rightarrow P_1 \rightarrow P_4$

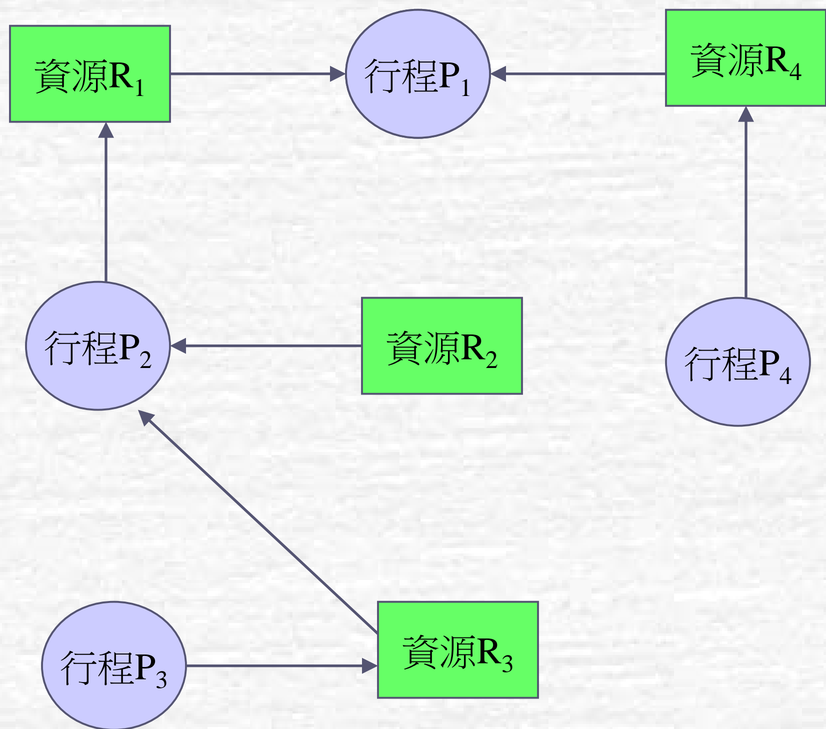
(a) P_5要求（3, 1, 1)後➔安全狀態，存在安全序列： $P_5 \rightarrow P_3 \rightarrow P_2 \rightarrow P_1 \rightarrow P_4$ ➔允許配置

(b) P_4要求（2, 1, 2)後➔不安全狀態，不存在安全序列➔不允許配置

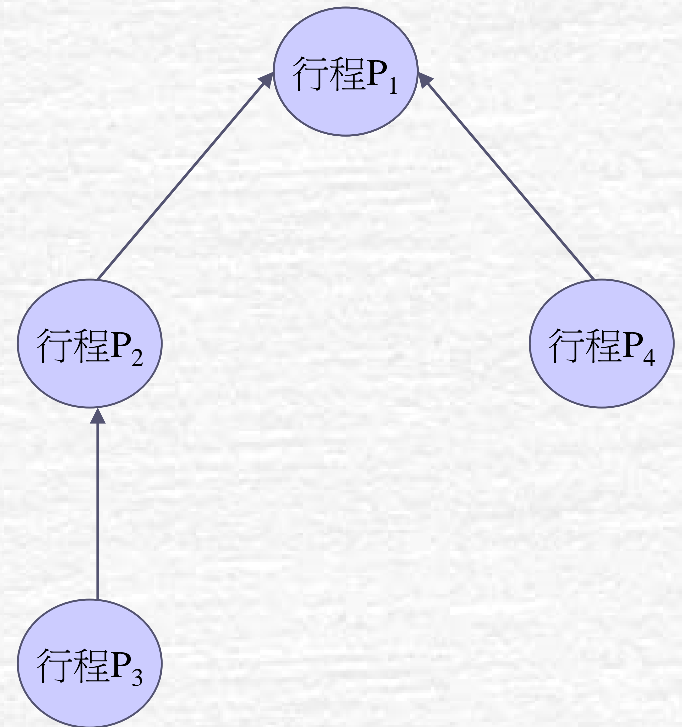
偵測死結

- 如果系統並沒有去預防或避免死結，也可以藉由提供死結的偵測與回復，來保障所有行程的正常執行
- 當系統中的每種資源只有一份時，我們可以利用等候圖來偵測死結
  - 等候圖是將資源配置圖的資源端點移除，而以Pi→Pj表示Pi正在等候Pj所擁有的某項資源
  - 當等候圖中的某些行程間形成一個封閉迴圈時，表示這些行程之間已經形成死結
- 如果每種資源的數量超過一個，則必須利用類似銀行家演算法中的安全演算法來作判斷

圖3-14 資源配置圖與等候圖



a. 資源配置圖



b. 等候圖

死結的解除

- 終止行程
  - 要終止一個行程或所有行程？
  - 要終止哪些行程？
- 回收資源
  - 要從哪些行程回收？
  - 如何回溯：在回收行程資源之時，必須將行程回溯到之前某個安全的狀態，以確定之後的執行能夠正確
  - 如何避免行程飢餓？

鴕鳥演算法

- 不論是死結的預防、避免、或偵測與回復機制，都是相當複雜而費時的
- 事實上，我們還沒有發現有任何很好的死結處理方法
- 對大多數作業系統而言，死結的情況其實很少發生
- 大多數的作業系統目前都直接忽略掉死結這個問題，假裝系統從來沒有發生過死結