

Title: NTU IM Operating-System HW 04

Student ID: R12631070

Name: 林育新

問答題

作業題目：第 7.3 題

圖 7.4 中所顯示的程式範例不一定會導致死鎖。

請說明 CPU 排程器 (scheduler) 在其中扮演的角色，以及甚麼情況下，會導致此程式發生死鎖。

```
/* thread_one runs in this function */
void *do_work_one(void *param)
{
    pthread_mutex_lock(&first_mutex);
    pthread_mutex_lock(&second_mutex);
    /**
     * Do some work
     */
    pthread_mutex_unlock(&second_mutex);
    pthread_mutex_unlock(&first_mutex);

    pthread_exit(0);
}

/* thread_two runs in this function */
void *do_work_two(void *param)
{
    pthread_mutex_lock(&second_mutex);
    pthread_mutex_lock(&first_mutex);
    /**
     * Do some work
     */
    pthread_mutex_unlock(&first_mutex);
    pthread_mutex_unlock(&second_mutex);

    pthread_exit(0);
}
```

作業回答

CPU 排程器的角色

CPU 排程器決定執行緒的執行順序與會不會被中斷。在本題中，如果排程器讓某個執行緒在只取得一把鎖的情況下就被中斷(可能是由於RR，或是更高優先級程式)，而另一個執行緒也在這時取得另一把鎖，則就有可能進入互相等待的狀態，產生死鎖。

✗ 發生死鎖的情況

在圖 7.4 中，`thread_one` 與 `thread_two` 兩個執行緒分別鎖定的 `mutex` 順序不同：

- `thread_one`：先鎖定 `first_mutex`，再鎖定 `second_mutex`
- `thread_two`：先鎖定 `second_mutex`，再鎖定 `first_mutex`

以下為可能造成死鎖的執行順序範例：

1. `thread_one` 先取得 `first_mutex`。
2. CPU 將控制權切換至 `thread_two`。
3. `thread_two` 取得 `second_mutex`。
4. `thread_two` 試圖鎖定 `first_mutex`，但因為已被 `thread_one` 鎖定，因此進入等待。
5. CPU 又切回 `thread_one`，它此時嘗試鎖定 `second_mutex`，但也被 `thread_two` 鎖定，進入等待。

此時兩個執行緒彼此互相等待對方釋放鎖，導致死鎖。

✓ 結論

圖 7.4 的程式範例**不一定**會導致死鎖，因為是否發生死鎖，取決於執行緒被排程的順序。如果排程器讓其中一個執行緒完整地取得兩把鎖並釋放，就不會發生死鎖；但若兩個執行緒分別持有一把鎖並等待對方的鎖，就會產生死鎖。

■ 作業題目：第 7.6 題

在真實的電腦系統中，無論是可用資源的數量，或是進程對資源的需求，長期下來（如幾個月）都不可能維持不變。資源可能會損壞或替換、新的進程會加入，新的資源也會被購買並加進系統。

若死鎖是由銀行家演算法（**Banker's Algorithm**）來控制，以下哪些變更可以在不導致死鎖的前提下**安全地執行**？請說明在什麼情況下這些變更是安全的。


選項如下：

- a. 增加 Available（加入新的資源）。
- b. 減少 Available（從系統中永久移除資源）。
- c. 增加某個進程的 Max（進程需要或想要更多資源）。
- d. 減少某個進程的 Max（進程決定其實不需要那麼多資源）。
- e. 增加進程的數量。
- f. 減少進程的數量。


📄 作業回答

✓ 相對安全的變更


1. a. 增加 Available (加入新的資源)

-  安全：增加可用資源只會讓系統的總資源數變多，銀行家演算法會在新的安全狀態下進行判斷，因此不會造成死鎖。


2. d. 減少某個進程的 Max (進程決定其實不需要那麼多資源)

-  安全：這會減少該進程未來可能的資源需求，讓系統更容易保持在安全狀態，反而降低發生死鎖的機率。

3. f. 減少進程的數量


-  安全：移除進程也就同時移除了對資源的需求，會釋放佔用的資源，有助於系統維持或轉回安全狀態。

補充說明：資源分配異常現象 (Anomaly)


 雖然減少進程數量或 Max 通常會讓系統看起來更容易安全，但在某些特殊情況下，這樣的變更可能會改變資源釋放的時機與順序，導致銀行家演算法的安全性檢查結果反而變差。因此，每次進行這類變更後，都必須重新檢查系統是否仍處於安全狀態，才能確保不會發生死鎖。

危險的變更 (可能導致死鎖)


1. b. 減少 Available (永久移除系統資源)

-  危險：可能使目前的系統狀態變得不安全，若剩餘的資源不足以滿足其他進程的需求，會導致死鎖。

2. c. 增加某個進程的 Max (進程想要更多資源)

-  危險：會增加進程的最大需求，可能導致系統從原本的安全狀態轉為不安全，導致無法保證所有進程都能順利完成。

3. e. 增加進程的數量

-  危險：增加新的進程就可能新增對資源的需求，若沒有即時調整系統資源或資源分配策略，就可能進入不安全狀態。


作業題目：第 7.12 題

給定一個系統快照 (Snapshot) 如下，請使用銀行家演算法判斷下列兩種 Available 狀態中，哪些為「不安全狀態 (Unsafe State) 」。

若為安全狀態，請列出可能完成進程的順序；若為不安全狀態，請說明原因。

Process	Allocation	Max
	A B C D	A B C D
P ₀	3 0 1 4	5 1 1 7
P ₁	2 2 1 0	3 2 1 1
P ₂	3 1 2 1	3 3 2 1


Process	Allocation	Max
P ₃	0 5 1 0	4 6 1 2
P ₄	4 2 1 2	6 3 2 5

 需要先計算每個進程的「需求 (Need) 矩陣」：

$$\text{Need} = \text{Max} - \text{Allocation}$$

Process	Need (A B C D)
P ₀	2 1 0 3
P ₁	1 0 0 1
P ₂	0 2 0 0
P ₃	4 1 0 2
P ₄	2 1 1 3

 小題 (a) : Available = (0, 3, 0, 1)

 嘗試找可完成的進程：

1. **P₁** : 需求 (1,0,0,1) ≤ Available (0,3,0,1) → **X** 不可完成
2. **P₂** : 需求 (0,2,0,0) ≤ Available →  可完成
 - 執行 P₂ · 釋放 Allocation (3,1,2,1) · 新的 Available = (0,3,0,1) + (3,1,2,1) = (3,4,2,2)
3. 接下來：
 - P₁ : 需求 (1,0,0,1) ≤ (3,4,2,2) →  執行後 Available = (3,4,2,2) + (2,2,1,0) = (5,6,3,2)
 - P₀ : 需求 (2,1,0,3) ≤ (5,6,3,2) →  執行後 Available = (5,6,3,2) + (3,0,1,4) = (8,6,4,6)
 - P₄ : 需求 (2,1,1,3) ≤ (8,6,4,6) →  執行後 Available = (8,6,4,6) + (4,2,1,2) = (12,8,5,8)
 - P₃ : 需求 (4,1,0,2) ≤ (12,8,5,8) →  執行

 結論：

安全狀態 (Safe)

一個可能完成的順序為：**P₂ → P₁ → P₀ → P₄ → P₃**

 小題 (b) : Available = (1, 0, 0, 2)

 嘗試找可完成的進程：

1. **P₁** : Need (1,0,0,1) ≤ Available (1,0,0,2) →  可完成
 - 執行 P₁ · 釋放 Allocation (2,2,1,0) · Available = (1,0,0,2) + (2,2,1,0) = (3,2,1,2)
2. 接下來：



- P_2 : Need $(0,2,0,0) \leq (3,2,1,2) \rightarrow$  執行 · Available = $(3,2,1,2) + (3,1,2,1) = (6,3,3,3)$
- P_0 : Need $(2,1,0,3) \leq (6,3,3,3) \rightarrow$  執行 · Available = $(6,3,3,3) + (3,0,1,4) = (9,3,4,7)$
- P_4 : Need $(2,1,1,3) \leq (9,3,4,7) \rightarrow$  執行 · Available = $(9,3,4,7) + (4,2,1,2) = (13,5,5,9)$
- P_3 : Need $(4,1,0,2) \leq (13,5,5,9) \rightarrow$  執行

 結論：

安全狀態 (Safe)

一個可能完成的順序為： $P_1 \rightarrow P_2 \rightarrow P_0 \rightarrow P_4 \rightarrow P_3$

最終結論

小題	狀態	執行順序
a	 安全	$P_2 \rightarrow P_1 \rightarrow P_0 \rightarrow P_4 \rightarrow P_3$
b	 安全	$P_1 \rightarrow P_2 \rightarrow P_0 \rightarrow P_4 \rightarrow P_3$