OS Homework 4

生機四 詹育晟 B10605023

7.3

A:

在這個範例中，兩條執行緒對 first_mutex 和 second_mutex 的加鎖順序正好相反，而作業系統的排程器（scheduler）決定了它們何時被搶走或重新分配 CPU 時間。假設 scheduler 在 thread_one 拿到 first_mutex 之後立刻切換到 thread_two，讓它先取得 second_mutex；接著兩者又分別嘗試鎖定對方已持有的 mutex，就會互相等待而發生死結。

相反地，如果 scheduler 讓某一條執行緒從頭到尾不被中斷──先依序鎖住兩把 mutex、執行完臨界區，再一次性全部解鎖──那麼另一條執行緒就能順利取得兩把鎖，不會有死結問題。由此可見，死結是否產生，關鍵就在於 scheduler 是否在「已鎖第一把但尚未鎖第二把」的時機點切換執行緒；若排程器不在這個關鍵時刻切換，程式反而會按部就班地完成鎖定與解鎖，不會陷入死結。

7.6

A:

a. 增加 Available

- **結論**：新增可用資源永遠是安全的。
- **原因**：
    1. **資料不受影響**：只有 Available 變大，Allocation 與 Max 都沒動，因此 Need (= Max–Allocation) 仍維持不變。
    2. **安全性單調**：原本若安全，新增資源後的安全序列依舊可用；即便原先不安全，也可能因資源增多而變成安全。

b. 減少 Available

- **結論**：刪減可用資源需謹慎，不一定安全。
- **原因**：
    1. **安全測試更嚴苛**：Work＝Available 初值縮水，Need≤Work 更難通過，原安全序列可能失效。
    2. **須重新驗證**：移除前必須確認要拿掉的資源實際在 Available 裡，移除後再跑一次 Safety Algorithm，確定仍有安全序列。

c. 增加某進程的 Max

- **結論**：擴大某進程最大需求不見得安全，必須重新檢查。
- **原因**：
    1. **通過判斷更困難**：Work 初值沒變，但 Need 變大，Need≤Work 條件更難符合，原安全序列或會失效。
    2. **雙重檢查**：更新後必須符合新 Max≥Allocation，並把新的 Max/Need 帶入演算法驗證安全性。

d. 減少某進程的 Max

- **結論**：縮小最大需求通常是安全的，只要新 Max ≥ 該進程已分配量。
- **原因**：
    1. **Safety 條件更好通過**：Work 初值不變，而 Need 變少，Need≤Work 更容易成立，原有安全序列不受影響。
    2. **合法後直接安全**：只要確認新 Max 合法（≥Allocation），就可認定安全，無須再跑一遍 Safety Algorithm。

e. 新增進程

- **結論**：增加進程前要驗證，否則可能破壞安全性。
- **原因**：
    1. **需求矩陣擴大**：多了一列 Max/Need（起始 Allocation＝0），系統整體需求提高。
    2. **Work 未增加**：Available 不變，但要滿足的進程更多，Need≤Work 條件更難全部符合。

f. 移除進程

- **結論**：刪掉進程肯定安全。
- **原因**：
    1. **資源回收**：該進程結束時會歸還所有 Allocation，Available 自動增加，同時從矩陣中移除該列。
    2. **Safety 條件更輕鬆**：Available 增多、進程數減少，Need≤Work 的檢查更容易全部通過。

7.12

7.12

(a) 用 Allocation 和 Max，由 Request = Max − Allocation，推出完整的表

Process	Allocation				Max				Request			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	3	0	1	4	5	1	1	7	2	1	0	3
P_1	2	2	1	0	3	2	1	1	1	0	0	1
P_2	3	1	2	1	3	3	2	1	0	2	0	0
P_3	0	5	1	0	4	6	1	2	4	1	0	2
P_4	4	2	1	2	6	3	2	5	2	1	1	3

(a) Available = (0,3,0,1) ⇒ 不安全

1. 初始化：work ← (0,3,0,1)

2. 第一輪掃描：找到唯一 Request ≤ work 之行程 ⇒ P_2 = (0,2,0,0)，執行後，釋出 Allocation to work = (0,2,0,1) + (3,1,2,1) = (3,4,2,2)

3. 第二輪掃描：剩下的 P_0、P_1、P_3、P_4 中，只有 P_1 能跑，work = (4,2,2) + (2,2,1,0) = (3,6,3,2)

4. 第三輪掃描：執行 P_3，work = (5,6,3,2) + (0,5,1,0) = (5,11,4,2)

5. 剩下的 P_0、P_4，的 D 需要 3，但是 work D = 2，因此無法繼續

⇒ 不安全#

(b) Available = (1,0,0,2) ⇒ 安全

1. work ← (1,0,0,2)
2. 執行 P_1 ⇒ work = (1,0,0,2) + (2,2,1,0) = (3,2,1,2)
3. 執行 P_2 ⇒ work = (3,2,1,2) + (3,1,2,1) = (6,3,3,3)

4. 依序執行 P_3 ⇒ work = (6,3,3,3) + (0,5,1,0)
= (6,8,4,3)

5. 執行 P_4 ⇒ work = (6,8,4,3) + (4,2,1,2)
= (10,10,5,5)

6. 執行 P_0 ⇒ work = (10,10,5,5) + (3,0,1,4)
= (13,10,6,9)

順序：$P_1 \to P_2 \to P_3 \to P_4 \to P_0$

⇒安全#