7.6

(a) 可以，鎖死的原因通常是系統資源不足，而增加可用資源可以解決這個問題。

(b) 不可以，減少可用資源可能會導致系統中某些進程無法獲得所需資源而陷入鎖死狀態。

(c) 某些情況可以，增加一個進程的最大需求量會使進程要求的資源更多，可能導致系統資源不夠分配給該進程，導致鎖死狀態；反之不會影響。

(d) 可以，減少一個進程的最大需求量會使系統資源更好地分配，因為進程要求的資源比原本來的小，系統資源能更有彈性的分配。

(e) 如果系統資源能被分配給新進程，且系統沒有進入不安全狀態，是可行的。

(f) 可以移除已釋放資源的進程，增加系統可用資源，對其他進程的分配更有彈性。

7.12

(a)

進程順序為：$P_2 \rightarrow P_1 \rightarrow P_3 \rightarrow$ unsafe，系統狀態為 unsafe，跑完 P3 後資源 D 沒有任何進程的資源 D 小於 Available 資源 D。

	Allocation				Max				Need				Available			
	A	B	C	D	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	0	3	0	1
P_1	2	2	1	0	3	2	1	1	1	0	0	1	3	4	2	2
P_2	3	1	2	1	3	3	2	1	0	2	0	0	5	6	3	2
P_3	0	5	1	0	4	6	1	2	4	1	0	2	5	11	4	2
P_4	4	2	1	2	6	3	2	5	2	1	1	3				

(b)

系統狀態為 safe，進程順序為：$P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0$

	Allocation				Max				Need				Available			
	A	B	C	D	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	1	0	0	2
P_1	2	2	1	0	3	2	1	1	1	0	0	1	3	2	1	2
P_2	3	1	2	1	3	3	2	1	0	2	0	0	6	3	3	3
P_3	0	5	1	0	4	6	1	2	4	1	0	2	6	8	4	3
P_4	4	2	1	2	6	3	2	5	2	1	1	3	10	10	5	5
													13	10	6	9

7.15

```
Mutex bridge_lock
Condition cond_north
Condition cond_south
Integer southbound_passing = 0
Integer northbound_passing = 0
```

```
Process SouthboundFarmer {
    lock(bridge_lock);
    while northbound_passing > 0 do
        wait(cond_north, mutex);
    end while
    ++southbound_passing;
    unlock(bridge_lock);
    // passing the bridge
    lock(bridge_lock);
    --sourthbound_passing;
    unlock(bridge_lock);
    broadcast(cond_south);
}
```

```
Process NorthboundFarmer {
    lock(bridge_lock);
    while southbound_passing > 0 do
        wait(cond_south, mutex);
    end while
    ++northbound_passing;
    unlock(bridge_lock);
    // passing the bridge
    lock(bridge_lock);
    --northbound_passing;
    unlock(bridge_lock);
    broadcast(cond_north);
}
```

8.5

(a)

連續內存分配：容易產生外部碎片，因為空閒與已分配區域相互交替且大小不一，當沒有連續的空閒區域可用時，將無法滿足大型進程的內存需求。

純分段和純分頁：不容易產生外部碎片，在內存中每個段或頁單獨分配一個空間，這樣空閒與已分配區域就不會相互交替。

(b)

連續內存分配：容易產生內部碎片，因為必須將進程分配到連續的物理內存區域中，當進程大小不是物理內存區域大小的倍數時，可能會浪費部分空間。

純分段和純分頁：不容易產生內部碎片，因為每段或每頁的大小都是固定的，並且進程將只使用它需要的段或頁，而不是整個連續內存區域。

(c)

連續內存分配：可以很容易地實現代碼共享，因為相同的代碼可以映射到不同的進程中的相同位置，並且可以通過修改某些頁表實現共享。

純分段和純分頁：需要額外的支持和操作來實現代碼共享。在純分段中，可以通過額外的段表實現共享；在純分頁中，可以通過將相同的頁映射到不同進程的相同位置來實現共享。

8.9

在分頁中，虛擬地址可以被分解為 page number 和 offset，這樣只需要一個頁表就能夠實現虛擬地址到物理地址的轉換。

在分段中，虛擬地址要被分解為段號和段內位移，因此需要維護多個段表，每個表都對應著一個段，這樣才能實現虛擬地址到物理地址的轉換。

分頁比分段需要更少的記憶體量。

8.13

(a)

Entries number = $2^{21} / 2^{11} = 2^{(21 - 11)} = 2^{10} = 1024$ entries.

(b)

Number of entries = $2^{16} / 2^{11} = 2^{(16 - 11)} = 2^5 = 32$ entries.

9.6

EAT = (1 - p) * 100 + p * (0.3 * 8000000 + 0.7 * 20000000) = 200

100 – 100p + 2400000p + 14000000p = 200

16399900p = 100

p = 100 / 16399900

9.8

(a) LRU replacement

Page fault	18																			
ref	7	2	3	1	2	5	3	4	6	7	7	1	0	5	4	6	2	3	0	1
f1	7	7	7	1	1	1	1	1	6	6	6	6	0	0	0	6	6	6	0	0
f2		2	2	2	2	2	5	5	5	7	7	7	7	5	5	5	2	2	2	1
f3			3	3	3	5	3	4	4	4	4	1	1	1	4	4	4	3	3	3

(b) FIFO replacement

Page fault	17																			
ref	7	2	3	1	2	5	3	4	6	7	7	1	0	5	4	6	2	3	0	1
f1	7	7	7	1	1	1	1	1	6	6	6	6	0	0	0	6	6	6	0	0
f2		2	2	2	2	5	5	5	5	7	7	7	7	5	5	5	2	2	2	1
f3			3	3	3	3	3	4	4	4	4	1	1	1	4	4	4	3	3	3

(c) Optimal replacement

Page fault	13																			
ref	7	2	3	1	2	5	3	4	6	7	7	1	0	5	4	6	2	3	0	1
f1	7	7	7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
f2		2	2	2	2	5	5	5	5	5	5	5	5	5	4	6	2	3	3	3
f3			3	3	3	3	3	4	6	7	7	7	0	0	0	0	0	0	0	0

9.17

(a)

計數器初始值 = 0

當頁面被訪問時，對應的頁面計數器加一

當需要進行頁面互換時，所有頁面計數器都減一

當要求的 page 不在 frame 中，最小頁面計數的頁面會被替換

(b)

Page fault	13																				
ref	1	2	3	4	5	3	4	1	6	7	8	7	8	9	7	8	9	5	4	5	4	2
f1	1	1	1	1	5	5	5	5	5	5	8	8	8	8	8	8	8	8	8	8	8	2
f2		2	2	2	2	2	2	1	1	1	1	1	1	9	9	9	9	9	9	9	9	9
f3			3	3	3	3	3	3	6	6	6	6	6	6	6	6	6	5	5	5	5	5
f4				4	4	4	4	4	4	7	7	7	7	7	7	7	7	7	4	4	4	4

(c)

Page fault	11																					
ref	1	2	3	4	5	3	4	1	6	7	8	7	8	9	7	8	9	5	4	5	4	2
f1	1	1	1	1	1	1	1	1	6	6	8	8	8	8	8	8	8	8	4	4	4	2
f2		2	2	2	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
f3			3	3	3	3	3	3	3	7	7	7	7	7	7	7	7	7	7	7	7	7
f4				4	4	4	4	4	4	4	4	4	4	9	9	9	9	9	9	9	9	6

9.19

系統在處理器和存儲器之間不斷進行無意義的頁面交換操作稱為 thrashing，會導致系統效能急劇下降的情況。

系統偵測 thrashing 通常以高缺頁率(page faults)或 CPU 利用率下降。

一旦系統檢測到繁忙交換，它可以採取增加物理內存、優化頁面置換算法、調整進程數量、優化程序內存使用、提高磁盤性能和增加頁面文件大小等來消除繁忙交換問題。