陳彥博 陳彥博 110034813

7-6

(a) 增加系統可派發的資源可以減少死結產的的機率，因為不會產生process所需的資源在別的process手上，且同時系統沒有足夠資源給任一process完成工作

(b) 減少最大工作量可以減少死結的機率，系統可以在process完成工作後，馬上取得該資源並派發給下一個process

(f) 減少process的數量也可以防止太多process向系統提出資源申請，導致沒有足夠資源派發而行成死結的情況

7-13

(a)

	Allocation	Max	Available	Need
P0	2001	4212	3321	2211 ✓
P1	3121	5252		2131
P2	2103	2316		0213
P3	1312	1424		0112 ✓
P4	1432	3665		2233

P0 → P3 → P1 / P2 / P4 順序任意

Work 3321 safe state

5322

6634

(b)

	Allocation	Need	Available
P0	2001	2211	2221 ✓
P1	4221	1031	
P2	2103	0213	
P3	1312	0112 ✓	
P4	1432	2233	

Work: 2221

4222

4334

↓

可滿足剩下3個

給過

(c)

	Allocation	Need	Available
P0	2001	2211	3301
P1	3121	2131	
P2	2103	0213	
P3	1312	0112	
P4	1432	2233	

Work: 3301

不給過，C資源不足以提供給剩下任一process

7-15.

```
semaphore mutex = 1

farmer A
[ sem_wait(&mutex)        ← 上鎖
  crossthebridge()        ← 過橋的行為
  sem_post(&mutex)        ← 解鎖

farmer B
[ sem_wait(&mutex)
  crossthebridge()
  sem_post(&mutex)
```

8.1

internal fragment：process 被派發的記憶體量比其所需多，同時多餘的空間也不能被其它 process 所使用，進而導致資源浪費

external fragment：process 與 process 間有零碎未被使用的記憶體空間，該空間可能因為太小而無法派發給其它程式所使用，進而導致資源浪費

8.9

paging 需要較多空間紀錄每個 page 所對應的 frame，因為一整個 process 會以一個固定大小的單位下去切分，這會造成有相當多的 page 需要定位到 frame，相對的，由於 segmentation 只把整個 process 切成幾個部份，如：Main, data, subroutine，且只需紀錄每個 segment 的 base address 與 limit，因此在紀錄此 Table 所需的記憶體空間較小

8.16

(a)

$2^{32} \div 2^{12} = 2^{20}$ entries

(b)

$512MB = 2^{29}$

$2^{29} \div 2^{12} = 2^{17}$ entries

陳彥博 110054813

9.8 = 7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0, 1

(a)	7	7	7	1		1	3	3	3	7		7	7	5	5	5	2	2	2	1
LRU		2	2	2		2	2	4	4	4		1	1	1	4	4	4	3	3	3
			3	3		5	5	5	6	6		6	0	0	0	6	6	6	0	0

18 次

(b)	7	7	7	1		1	1	6	6		6	0	0	0	6	6	6	0	0	
		2	2	2		5	5	5	7		7	7	5	5	5	2	2	2	1	
FIFO			3	3		3	4	4	4		1	1	1	4	4	4	3	3	3	

17 次

(c)	7	7	7	1		1	1	1			1		1	1	1	1				
		2	2	2		5	5	5			5		4	6	2	3				
OPT			3	3		3	4	6	7		0		0	0	0	0				

11 次

9.11 當整個程式執使用的 page 很多時，(也就是把程式分段後，某個 page 的出現幾乎永遠是最後時)，則 LRU 效果較好，如 1, 2, 3, 4, 2, 3, 4, 1, 2, 3, 4, 但若是程式所需的 page 很不多時，(某個 page 則開始常出現，後來就沒出現過了)，則用 LRU 比較好，如：1, 1, 1, 1, 1, 2, 3, 4, 2, 3, 4, 5, 2, 3, 4

9.17

(a)
(i) initial value = 0
(ii) 當一個 page 被丟到 frame 時，且該 page 未曾被紀錄在該 frame 過，counter +1
(iii) 當原本有被紀錄在該 frame 上的 page 不再被參考時，-1 (移出時計算)
(iv) 挑出 counter 數最小的 frame 來 replace，當相同 counter，FIFO

(b)

1	2	3	4	5	3	4	1	6	7	8	7	8	9	7	8	9	5	4	5	4	2
1①	1①	1①	1①	5②	5②	5②	5②	5②	5②	5②	5②	8②	8②	8②	8①						8②
	2①	2①	2①	2①	2①	2①	1②	1②	1②	1②	1②	1②	1②	1②	5②	5②	5②				5②
		3①	3①	3①	3①	4②	4②	6②	6②	8②	8②	8②	9②	9②	9②	9②	9②				9②
			4①	4①	4①	4①	4①	4①	7②	7②	7②	7②	7②	7②	10②	10②	4②				2②

14 Page Fault

(c)

1	1	1	1	1		6	6	8		8			8			8				
	2	2	2	5		5	5	5		5			5			2				
		3	3	3		3	7	7		7			4			4				
			4	4		4	4	4		9			9			9				

11 Page Fault

9.19

Thrashing 發生的原因為：當發生 page fault 時，要從 process 去搶其它 process 的 frame，而其它 process 也會搶被走的 frame，在 OS 沒有分派足夠 frame 的情況下，CPU 一直在處 page fault error 而沒有實際產能，而此的排班程式就為 CPU 太閒，可能會再繼續派發更多工作給 CPU，造成惡性

CPU 可以透過此對系統的 multiprogramming degree 與 CPU 的利用率來觀察是否有 Thrashing（CPU utilization 低，but multiprogramming degree 高，代表有 Thrashing）

CPU 可以減少 multiprogramming degree，並移除 process 移出（CPU 來解決此問題）