116

Operating Systems, 2nd Term Exam. Range: Chapter 7, 8, 10. June 8, 2020

[redacted]

CLOSE BOOK EXAM. 130 points.

Write your answers on the exam sheets directly.

1. Write down the original terminology of the following abbreviations: (5%)
   1) MMU memory management unit
   2) TLB translation look-aside buffer
   3) DLL (in 9.1.5) dynamic linking library
   4) LRU (in page replacement) least recently used
   5) NUMA (memory) non-uniform memory access

2. Explain the following terminology: (40%)
   1) Relocation register (in memory protection)
      用來儲存 base 的 register。將 virtual address 轉換成 physical address 的過程中，要讀 relocation register 得到 base 值，再 base+virtual address 得到 physical address。
   2) Limit register (in memory protection)
      代表該 process 可用的位址範圍，limit 值與 virtual address 的 offset 比較，判斷 offset 是否超界使用到別的 process 的記憶體。
   3) dynamic loading:
      程式執行時，只載入主程式到記憶體，其他 routine 則等到被呼叫時再載入。
   4) external fragmentation:
      滿足空間需求的非連續記憶體空間。
   5) working set:
      最近的 n 個 page reference 中，所有用到的 page 的集合。
   6) demand paging
      不從 backing store 載入所有可能用到的 page 到記憶體，而是等到用到時(通常會產生 page fault)，才載入 page。
   7) Reentrant code
      可以重複使用，也可以與其他 process 共享的 non-self-modifying code。
   8) Thrashing:
      process 忙於做 paging，花在 paging 的時間比實際執行還多，這種情況稱 thrashing。
   9) Load sharing

-3

-3

10) rollback

deadlock 發生時，回復到 safe state 的過程。

11) memory stall

存取記憶體時，因CPU與記憶體的工作頻率不同，需要等待記憶體回應的情況。

12) dirty bit

又稱 modify bit，在 page replacement 時，判斷 page 是否有被修改過，以用來決定該 page 是否要寫入 backing store (還是要直接覆寫 page)。

13) reference bit

判斷 page 是否有被 reference 過，若 reference bit 一直為 0 (沒有被 reference)，就把它取代為別的 page。

14) Belady's anomaly

在 FIFO algorithm 中，page fault 次數隨 frame number 值增加的情況。

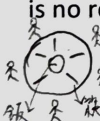
15) linker

付責連結程式碼與所使用的 library。

16) Relocatable code

不把位址寫死，經 relocation 後就可以在記憶體的任意位址上執行的 code。

17) The dining-philosophers problem (Hint: just describe the problem. Its solution is no required.)



有n位哲學、n枝筷子，哲學家需要2枝筷子才能吃飯。在這些條件下，使得有些哲學可以吃飯，有些要等待筷子的同步問題。

18) Compaction (in memory management)

把記憶體的 process 及可用空間碎片重新排列，排出連續的可用空間的機制。

19) Locality (of reference)

在一個時間點，process 所有用到的 page (包含 local variable 與 global variable)。

20) NUMA

多個CPU的系統上，因 memory 連接方式不同而產生 memory access 所需時間不同的情況，例如：部份 memory 透過 bus 連結，部份透過 ethernet 連結，兩者存取所花的時間就不一樣。

-1

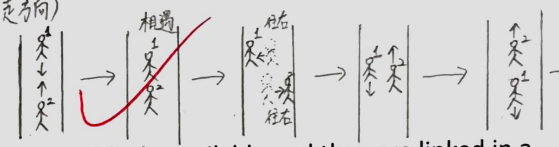
3. (3%) (1) What is 'livelock'? (Section 8.2.1)
   (3%)(2)Show an example of livelock.
   (4%)(3)show an approach to solve the livelock you shown in (2)

(1) 兩 process 或 thread 執行時因自己的選擇，造成互相衝突的情況。

(2) 以兩人走在一固定寬度的通道上的情況舉例，

兩人行走方向不同 → 相遇時，1往左讓，2往右讓（考慮他們的行走方向） → 過不了，1再往右讓，2再往左讓 → 還是不行，1再往左讓，2再往右讓 → 依舊過不了，一直這樣持續下去…

(3) 設定兩人遇到障礙時要往右讓（考慮行走方向）：



→ 解決 livelock

4. Suppose there are four free memory blocks available and they are linked in a FREE-MEMORY single link list. The sequence and the size of them are 15Kbytes, 20KBytes, 8Kbytes and 10Kbytes. Contiguous memory allocation is required. Suppose there comes a new process that requires 9Kbytes. Suppose we use the following allocating policies, block would be chosen and what the fragmentation would be.

   (3%)(1)FIRST fit

   15Kbytes block, fragmentation = 15-9 = 6Kbytes

   (3%)(2)BEST fit

   10Kbytes block, fragmentation = 10-9 = 1Kbytes

   (4%)(4)Suppose that is a new process that requires 21Kbytes. Suppose you are the designer of the operation system, what operation you would choose to handle this situation.

   把那21Kbytes 分成 20Kbytes 與 1Kbyte，再用 best fit 將 20Kbytes 放入 20Kbytes block，另外 1Kbyte 則放入 8Kbytes block。

5. (25%)
   (1)(3%)Show an example of deadlock.
   (2)(3%)Show an example of deadlock with mutex locks or other synchronization mechanism. (Hint: you can design such a deadlock by yourself. No need to follow the example in the textbook.)
   (3)(4%)suppose the operating system does not support any mechanism or policy for detecting deadlocks and breaking deadlocks. As a programmer, how would

know that your cooperating programs (or multi-threaded application program) is quite likely deadlocked.?

(4)(5%)Show an approach that can support deadlock prevention. Please use your example in (2).

(5) (10%)Suppose total system resources are: R1=6, R2=5, R3=7, R4=6. Available resources are: R1=3, R2=1, R3=1, R4=2. The processes currently holding resources:

	R1	R2	R3	R4
P1	1	2	2	1
P2	1	0	3	3
P3	1	2	1	0

寫在最後

Process maximum resource:

-5

	R1	R2	R3	R4
P1	3	3	2	2
P2	1	2	3	4
P3	1	3	5	0

Is the system currently in a safe or unsafe state? (Note that you must explain why you think it is safe or unsafe.)

第(2)題 →(2)

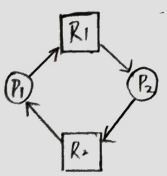
令現在程式中有lock1, lock2，且有transaction1()、transaction2()，pseudocode如下：

```
transaction1(){
    acquire(lock1);
    acquire(lock2);
    //do something
    release(lock2);
    release(lock1);
}

transaction2(){
    acquire(lock2);
    acquire(lock1);
    //do something
    release(lock1);
    release(lock2);
}
```

兩function同時開始執行，transaction1()先拿到lock1，而transaction2()先拿到lock2，但執行到第2行，transaction1()會等待lock2，transaction2()會等待lock1，兩者互相等待、互不相讓，形成deadlock。

第(1)題 →(1)



P1在等P2釋放R1，P2在等P1釋放R2
兩者互相等待、互不相讓，產生deadlock。

(3) 檢查工作管理員，看CPU使用率、RAM使用率。

(4) 給定lock1編號為100，lock2編號為200，讓transaction1(), transaction2()在取得lock時，必須從編號小的取到大的。若持有編號較大的lock，(ex: lock2) 想取得編號小的lock (ex: lock1) 的時候，必須先釋放lock。

這樣可以避免circular wait。

6. (3%)(1)What is the function of TLB?
(3%) (2)What would happen in a paging system if there is not a TLB?
(3%) (3)What is the motivation for "hierarchical paging"?
(3%) (4)What is the definition of 'linear address' in IA-32?
(3%) (5)What is a 'selector" and a 'descriptor" in IA-32?

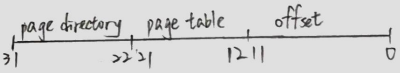
(1)用來儲存 reference 過的 page number 與 frame number 的高速 buffer，減少因為存取記憶體中的 page table，造成的等待時間。

(2)每次 paging 都要存取存在記憶體中的 page table，取得 frame number 後再至記憶體存取資料，造成效率降低。

(3)記憶體的位址很長、空間很大，讓 page table 變得很長，會浪費很多空間，用 hierarchical paging 減少 page table 佔用的空間，也加快存取速度。

-1

(4) logical address 經 segmentation unit 轉換而得到的 32-bit address，會再送入 paging unit 做轉換。格式：



(5) selector 是 descriptor table 的 index，用來找對應的 segment base 與 segment limit。

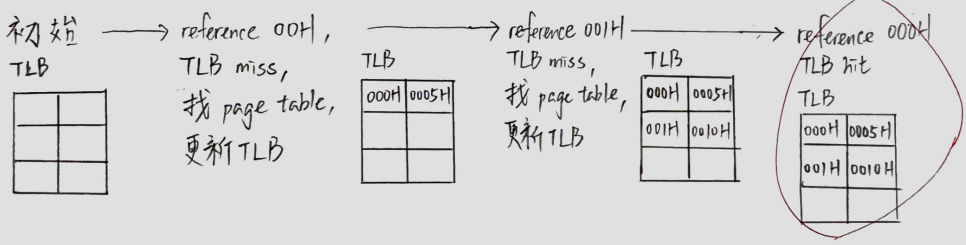
descriptor 是 descriptor table 的 element，其中的資訊包含 segment base、segment limit ... 等。

7. (10%)(1)Describe the operation of 'second chance algorithm".
(2) How does a programmer know one's program has got thrashing? How to reduce the occurrence of thrashing?

(1) 所有page連成circular queue，再循環檢查page的reference bit挑選page做replacement。檢查到reference bit=1時，重設為0，繼續檢查，若繞一圈回來發現該reference bit=0時，就取代成別的page。

(2) 記錄page fault frequency，當值超過一定上限時，認定為發生了thrashing。如何減少發生？1.最佳化程式 2.裝更多的記憶體。

8. (5%)Suppose that the TLB only stores the required information for the current running process. There are currently 3 pages active(present) in main memory. The are (1)the 000H page is in in 0005H frame. (2)the 001H page is in 0010 frame. (3)The 002H page is in 0008H frame. Suppose the pages entries for the TLB are stored as 000H, 001H and 002H accordingly. Draw a block diagram to show the current contents of TLB.



9. (10%)Consider the following page reference string: 1, 2, 3, 4, 7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6. (1)Assuming demand paging with four frames, how many page faults would occur for the LRU algorithm? (2)How many page faults would occur for the optimal algorithm?

(1)

	1	2	3	4	7	2	3	1	2	5	3	4	6	7	7	1	0	5	4	6
	1	1	1	1	7			7		5		5	5	7		7	7	7	4	4
		2	2	2	2			2		2		2	6	6		6	6	5	5	5
			3	3	3			3		3		3	3	3		1	1	1	1	6
				4	4			1		1		4	4	4		4	0	0	0	0

15 page faults

(2)

	1	2	3	4	7	2	3	1	2	5	3	4	6	7	7	1	0	5	4	6
	1	1	1	1	1					1		1	1				0		0	
		2	2	2	2					5		5	5				5		5	
			3	3	3					3		4	6				6		6	
				4	7					7		7	7				7		4	

10 page faults

(5) total : $R_1=6$, $R_2=5$, $R_3=7$, $R_4=6$

	Allocation				Max				Need				Available			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_1	1	2	2	1	3	3	2	2	2	1	0	1	3	1	1	2
P_2	1	0	3	3	1	2	3	4	0	2	0	1				
P_3	1	2	1	0	1	3	5	0	0	1	4	0				

P_1 : Need $=(2,1,0,1) \le (3,1,1,2) =$ Available, fit, safe sequence $=\{P_1\}$

Available $=(3,1,1,2)+(1,2,2,1)=(4,3,3,3)$

P_2 : $(0,2,0,1) \le (4,3,3,3)$, fit, safe sequence $=\{P_1, P_2\}$

Available $=(4,3,3,3)+(1,0,3,3)=(5,3,6,6)$

P_3 : $(0,1,4,0) \le (5,3,6,6)$, fit, safe sequence $=\{P_1, P_2, P_3\}$

Available $=(5,3,6,6)+(1,2,1,0)=(6,5,7,6)$

在 safe state，因為找得到 safe sequence $\{P_1, P_2, P_3\}$