

長庚大學106學年度第一學期作業系統期末測驗 (滿分108)

系級:

姓名:


學號:

1. (8%) Please (a) define “Race Condition” and (b) provide an example for Race Condition. You can use the case, counter ++ and counter -- are in two different processes, as the example. (Hint: the assembly code of counter ++ could be: $r_1 = \text{counter}$; $r_1 = r_1 + 1$; $\text{counter} = r_1$;)

Answer:

(a) (5%) A situation where the outcome of the execution depends on the particular order of process scheduling.

(b) (5%)

- One counter++ and one counter--
 $r_1 = \text{counter}$ $r_2 = \text{counter}$
 $r_1 = r_1 + 1$ $r_2 = r_2 - 1$
 $\text{counter} = r_1$ $\text{counter} = r_2$
- Initially, let $\text{counter} = 5$
 - P: $r_1 = \text{counter}$
 - P: $r_1 = r_1 + 1$
 - C: $r_2 = \text{counter}$
 - C: $r_2 = r_2 - 1$  A Race Condition!
 - P: $\text{counter} = r_1$
 - C: $\text{counter} = r_2 = 4$
- The result can be 4, 5 or 6

2. (10%) Let's consider the Readers and Writers Problem. We now have two mutex instances, mutex1 and mutex2. Please complete the code of writers and readers. (Hint: you should fill in all ? with mutex1 or mutex2 in the following sample code.)

Writer:

```
wait(?);  
... writing ... ;  
signal(?);
```

Reader:

```
wait(?);  
readcount++;  
if(readcount == 1) {wait(?);}  
signal(?);  
... reading ... ;  
wait(?);  
readcount--;  
if(readcount == 0) {signal(?);}  
signal(?);
```

Answer:

Writer:

```
wait(mutex1);  
... writing ... ;  
signal(mutex1);
```

Reader:

```
wait(mutex2);  
readcount++;  
if(readcount == 1) {wait(mutex1);}  
signal(mutex2);  
... reading ... ;  
wait(mutex2);  
readcount--;  
if(readcount == 0) {signal(mutex1);}  
signal(mutex2);
```

如果所有mutex1都換成mutex2，且所有mutex2都換成mutex1，也正確。

3. (8%) To manage deadlocks, OS can do “deadlock prevention” or “deadlock avoidance” to guarantee that there is no deadlock. If we allow deadlocks, OS can do “deadlock detection” and recover the system from deadlock states. Please define (a) deadlock prevention and (b) deadlock avoidance.

Answer:

(a) Prevent the necessary conditions (i.e., mutual exclusion, hold and wait, no preemption, and circular wait) of a deadlock

(b) Make sure that the system always stays at a “safe” state (by Banker’s Algorithm).

4. (12%) Banker’s Algorithm is a deadlock avoidance algorithm. There are 5 processes $\{P_0, P_1, P_2, P_3, P_4\}$ and three types of shared resources $\{A, B, C\}$ in the system, and the details are in the following table. Now, P_1 further has a request $(0, 1, 1)$ to use 1 more instance of B and 1 more instance of C. Should the request be granted? Please provide the reason to support your answer.

	Allocation			Max			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	7	4	3	2	3	2
P1	2	0	0	2	2	2	0	2	2			
P2	3	0	2	8	0	6	5	0	4			
P3	1	1	1	4	2	2	3	1	1			
P4	0	0	2	5	3	3	5	3	1			

Answer:

Yes.

1) $(0, 1, 1)$ does not exceed the Need $(0, 2, 2)$ of P_1 .

2) $(0, 1, 1)$ does not exceed the Available $(2, 3, 2)$ of the system.

3) After the system grants the request, we still can find out a safe sequence:

Available $(2, 3, 2) \rightarrow$ Available $(2, 2, 1)$.

P_1 has Need $(0, 1, 1)$ and Allocation $(2, 1, 1)$.

Now we run Banker’s Algorithm: Available $(2, 2, 1) \rightarrow P_1$ Need $(0, 1, 1) \rightarrow$ Available $(4, 3, 2) \rightarrow P_3$ Need $(3, 1, 1) \rightarrow$ Available $(5, 4, 3) \rightarrow P_4$ Need $(5, 3, 1) \rightarrow$ Available $(5, 4, 5) \rightarrow P_2$ Need $(5, 0, 4) \rightarrow$ Available $(8, 4, 7) \rightarrow P_0$ Need $(7, 4, 3)$.

5. (10%) Now, let’s do deadlock detection. There are 5 processes $\{P_0, P_1, P_2, P_3, P_4\}$ and three types of shared resources $\{A, B, C\}$ in the system, and the details are in the following table. Is the system in a deadlock state? If no, please provide a sequence to complete all processes. If yes, please still try to provide a sequence to complete as many processes as possible, and then show me which processes are in the deadlock.

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	2	0
P1	2	0	3	5	0	1			
P2	3	0	1	0	3	0			
P3	1	1	1	3	3	1			
P4	4	0	1	0	0	3			

Answer:

Yes, it is in a deadlock.

Available $(0, 2, 0) \rightarrow P_0$ Request $(0, 0, 0) \rightarrow$ Available $(0, 3, 0) \rightarrow P_2$ Request $(0, 3, 0) \rightarrow$ Available $(3,$

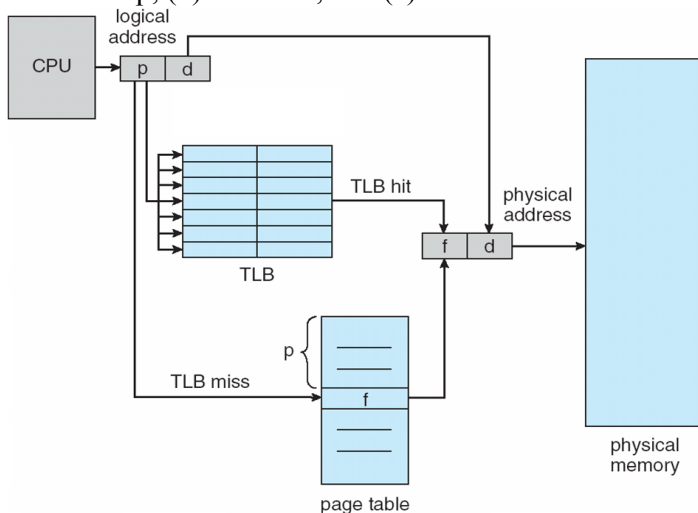
3, 1) → P₃ Request (3, 3, 1) → Available (4, 4, 2).
P₁ and P₄ can not be completed and are in the deadlock.

6. (12%) Please define (a) external fragmentation and (b) internal fragmentation of memory management. If we use “Paging” to manage memory, (c) is it possible to have external fragmentation? (d) is it possible to have internal fragmentation? You have to provide reasons for the answers of sub-questions c and d.

Answer:

- (a) External Fragmentation – total memory space exists to satisfy a request, but it is not contiguous
(b) Internal Fragmentation – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
(c) No. All memory is partitioned in a page size, and all requests are with a page unit.
(d) Yes. For example, if we need only 3 KB memory space, there will be a 1 KB internal fragmentation in an allocated 4-KB page.

7. (12%) For memory management with page tables, considering the following figure, please define (a) what is p, (b) what is f, and (c) what is d. Please also explain (d) what does TLB do.



Answer:

- (a) Page number
(b) Frame number
(c) Offset
(d) TLB is usually some on-chip SRAM and can keep the frame numbers of recently referred pages so as to reduce the extra latency for accessing page tables.

8. (12%) There is system with only 3 memory frames. Given a reference string of pages {5→2→0→1→0→1→3→4→5→1→3}, please illustrate the page replacement of (a) the Least-Recently-Used (LRU) algorithm and (b) the First-In-First-Out (FIFO) algorithm. You should show the contents of memory frames and the LRU and FIFO queues.

Answer:

(a)

Memory Frame

5	5	5	1	1	1	1	1	5	5	5
	2	2	2	2	2	3	3	3	1	1
		0	0	0	0	0	4	4	4	3

LRU Queue

5	2	0	1	0	1	3	4	5	1	3
	5	2	0	1	0	1	3	4	5	1
		5	2	2	2	0	1	3	4	5

(b)

Memory Frame

5	5	5	1	1	1	1	1	5	5	5
	2	2	2	2	2	3	3	3	1	1
		0	0	0	0	0	4	4	4	3

FIFO Queue

5	2	0	1	1	1	3	4	5	1	3
	5	2	0	0	0	1	3	4	5	1
		5	2	2	2	0	1	3	4	5

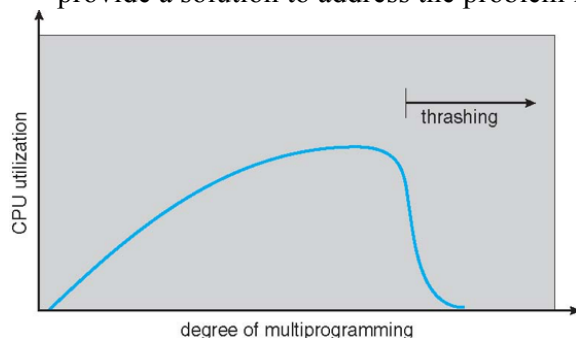
Queue的頭在最上面或是最下面都可以，只要內容正確即可，方向不拘。

Memory frames裡面的內容如果有不必要的移動會產生額外的page writes，會扣分。

9. (8%) The basic idea of Second-Chance page replacement algorithm is to approximate the behavior of Least-Recently-Used (LRU) page replacement algorithm. It keeps a reference bit for each page. When Second-Chance algorithm has to find a victim page, it sequentially tests all pages. If the reference bit of a page is 1, it changes the bit as 0 and goes to the next page. If the reference bit of a page is 0, the page is selected as the victim page. Please explain when will Second-Chance algorithm change a reference bit from 0 to 1.

Answer: When a page is referred (read or written), the corresponding reference bit will be set to 1.

10. (8%) In computer science, thrashing occurs when a computer's virtual memory subsystem is in a constant state of paging, rapidly exchanging data and binaries in memory for data and binaries on disk. (a) Please use the following figure to provide one possible reason for causing thrashing. (b) Please provide a solution to address the problem is question (a).



Answer:

(a) When there are too many page faults, the CPU utilization could be low. However, when the CPU utilization is low, the scheduler might launch more processes for their execution, and it makes the CPU utilization even lower because there are page faults. That is thrashing.

(b) Operating systems should monitor the page-fault rate. If the page-fault rate is too high, scheduler should not load any new user processes despite the low CPU utilization.

11. (8%) In the final project, we have used the following four commands. Please explain the purposes for using the four commands.

(a) *make*

- (b) *sudo insmod hello.ko*
- (c) *sudo rmmod hello*
- (d) *dmesg*

Answer:

- (a) Build the kernel module
- (b) Insert the kernel module
- (c) Remove the kernel module
- (d) Print the message buffer of the kernel.