SOLUTION

NATIONAL UNIVERSITY OF SINGAPORE

CS2106 – INTRODUCTION TO OPERATING SYSTEMS

(Semester 1: AY2017/18)

Time Allowed: 2 Hours

Questions 1-7: Indicate whether the given statement is **True / False**. Support your answer with brief explanation / counter example wherever appropriate. You may also state any assumptions if necessary. **Each question is worth 4 marks.**

1. On a system using pure paging-scheme, when we attach a **share memory region** to a process, the memory address returned by the **`shmat()`** function is likely to be at the boundary of a memory page.

ANS: TRUE. We need to set the translation (i.e. frame number) to the shared region different from other memory locations.

2. All threads in the **same process** can share the same **page table**.

ANS: TRUE. Even though threads has their own stack space, they are using the same virtual memory space.

3. The size of a Page Table Entry (PTE) is dependant on the total size of the logical / virtual memory space of a process.

ANS: FALSE. PTE stores frame number mainly, which is influenced directly by the size of the physical RAM not the size of the virtual memory space.

4. The size of a Direct Paging Page Table (i.e. the number of page table entries) of a Process is dependant on the total size of the logical / virtual memory space utilized by the process

ANS: FALSE. Page table has the same number of entries regardless of actual usage.

5. The Virtual Address (VA) can never be the same as the Physical Address (PA), i.e. there can be no instance where VA 0xZZZZ ZZZZ get translated to PA 0xZZZZ ZZZZ.

ANS: FALSE. It is possible for a Page to map to a frame with the same number, i.e. Page 123 to Frame 123 by chance.

6. Accessing the N^{th} block of a file managed by FAT scheme requires O(N) hard disk accesses in the worst case.

ANS: FALSE. FAT is an in-memory structure, so traversing the link requires zero hard disk accesses.

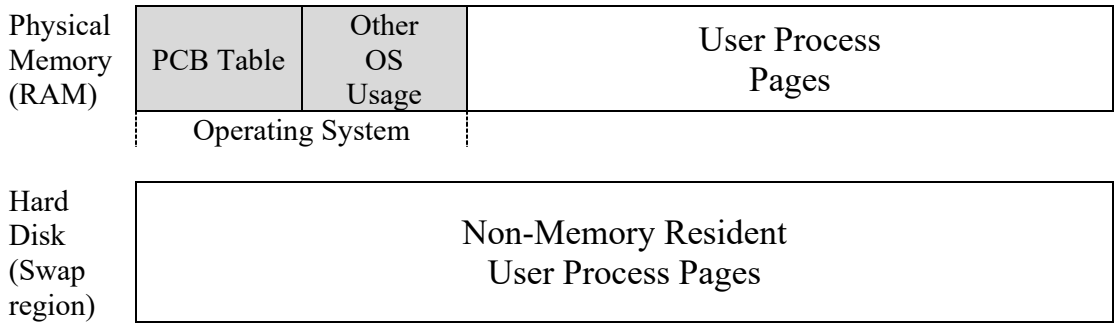
7. Suppose process A is current executing and is given **2 time units** time quantum to run:

0	1	2	3
A	**A**	…	…

In Unix, a **new Process B** can only arrive in the system during the **shaded region**. Note that ITI is not part of the shaded region.

ANS: TRUE. In unix, process can be created only via fork() / clone() mechanism ➔ A has to invoke one of these ➔ only when A is executing.

Questions 8 - 13: Below is an illustration of virtual memory organization on a certain machine. Note that the size of each region is not to scale (i.e. the size of each region is not accurately represented by size of the boxes). **Each question is worth 3 marks.**



For ease of reference, use the following short form for each of the regions:

a. **OS-P**: OS memory region for PCB tables.
b. **OS-O**: OS memory region for other usages.
c. **UPP**: User Process Pages in physical ram.
d. **SWAP**: Non-memory resident user process pages.
e. **NIL**: Not part of the virtual memory space.

For each of the following items, indicate the region(s) it can reside. State any assumptions if necessary.

8. Page table of a process. [a]
9. Open file table. [b]
10. Dynamically allocated linked list nodes in a program. [c, d]
11. File descriptor returned from an **`open(...)`** system call. [c, d]
12. The code for the Process Scheduler. [b]
13. The binary of a compiled program, e.g. **`a.out`**. [e]

---

14. **[6 marks]** Outline a simple program that can find out the page size of a machine. You should state clearly how to run the program and interpret the result.

ANS: Many possibilities.

Common approaches:

- Using timing (page fault / TLB fault incurs a much longer latency then normal memory access).
- Using segmentation fault. In pure paging scheme, seg.fault can only be detected by going beyond a page.

Sample answer:

- Create an array of large size.

- Access the item sequentially and monitor the time needed for each access.
- When there is a huge spike, it is likely we have hit the page boundary.

Common wrong / partially wrong answers:

- Assumed I/O buffer size is the same as page size and use the printf() example from tutorial to find the buffer size.
- Assumed that the memory address will have a "huge jump" between pages. This is a pretty serious misunderstanding as it confused **logical address (seen by the program)** and the **physical address (translated and used by hardware).** If you print the address of a variable inside the program (e.g. printf("%p", &myVar)), you only see the **logical address**.

15. **[9 marks]** Upon the termination of a process **P**, operating system is supposed to "clean up the resources used by **P**". Answer the following briefly:
    a. **[3 marks]** How is the OS notified of **P**'s termination?

    b. **[6 marks]** Indicate the resources to be cleaned up and how can the OS find out the relevant information to perform the cleaning. You should be as specific as possible and refer to ideas covered in this course.

ANS:
    a. Process will calls exit() system call implicitly or explicitly.
    b. Memory frames (using page table to find out), All opened filed (using file descriptors to find out)

16. **[9 marks]** Indicate whether the working set W(T, Δ) increases (more pages are needed), decreases (less pages are needed) or stable (no change in number of pages needed) for the following scenarios:
    a. **[3 marks]** We are executing a loop to calculate values to be stored in a **large array**.

    b. **[3 marks]** We are currently executing a **recursive function**.

    c. **[3 marks]** We are currently blocked on a critical section implemented by **busy waiting**.

    State any assumption if necessary.

ANS:

Answers depend strongly on your assumption and explanation. By default, we accept the following:
    a. Increases. Data pages for the array needs to be read in / write out.
    b. Increases. Stack space will increases.
    c. Stable. Code executed is the same (busy waiting is usually an infinite loop).

Alternative answers with assumptions:
    - If we look at a small enough time window, then it can be argued that (a) and (b) has "stable" number of pages. e.g. if the time window covers only one iteration of loop in (a), then we only access a single element (from one page) every window. Similar argument can be made for (b).

17. **[12 marks]** Given the following two code fragments:

Code One	**Code Two**

`Wait( S1 )`
`A`
`B`
`Signal( S1 )`
	`Wait( <X> )`
	`C`
	`D`
	`Signal( <X> )`

Suppose each code is executed by many concurrent tasks, we want to examine whether the following execution sequence:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	C	B	D	C	A	D	B	A	A	C	B	B	C	D	D

is possible under the following scenario:

a. <X> is S1 (i.e. same semaphore used in Code One). S1 is a binary semaphore initialized to 1.

b. <X> is S1. S1 is a general semaphore initialized to 2.

c. <X> is S2 (i.e. a different semaphore). Both S1 and S2 are binary semaphores initialized to 1.

d. <X> is S2. Both S1 and S2 are general semaphores initialized to 2.

For each scenario, indicate the **earliest (**i.e. **leftmost)** sequence number where impossible execution is detected, e.g. if you think D (at 4) is impossible due to the semaphore restriction, indicate "4" in the answer booklet. Briefly explain your reasoning.

If there is nothing wrong with the sequence, just indicate **0** in the answer. There is no need to explain for such case(s).

ANS:

a. 2. Need to signal S1, before C can execute.
b. 11. Need to signal S1, before C can execute.
c. 10. Need to signal before the second copy of Code One can execute.
d. 0

18. **[18 marks]** Below is the File Allocation Table (**FAT**) and the directory entry of the **root directory** on a mysterious disk:

0	2	8	11
1	0	9	14
2	15	10	BAD
3	EOF	11	4
4	EOF	12	EOF
5	13	13	7
6	9	14	3
7	12	15	5

FAT (16 disk blocks)

easy.txt	6
exam.c	1
kidding.bmp	8

Directory Entry (for **root**)

a. **[2 marks]** Give the file size for the file "**easy.txt**" if each disk block is 1KB.

b. **[5 marks]** Convert the file "**exam.c**" into I-Node representation. Note the following:
   - Use the same disk block numbers in the same sequence currently occupied by "**exam.c**".
   - The I-Node structure is defined as:
     - 3 direct data block pointers
     - 1 indirect data block pointer (pointing to 3 direct pointers)
     - 1 double indirect pointer (pointing to 3 indirect pointers)
     - 1 triple indirect pointer (pointing to 3 double indirect pointers)

   Show the I-Node for "**exam.c**".

c. **[5 marks]** Perform a **defragmentation** on the disk. So that:
   - Files uses consecutive blocks.
   - Free space (if any) are at the end of the disk.
   - The order of the files is the same as the directory entries (i.e. "**easy.txt**" followed by "**exam.c**" followed by "**kidding.bmp**").

   Show the FAT and directory entries after defragmentation.

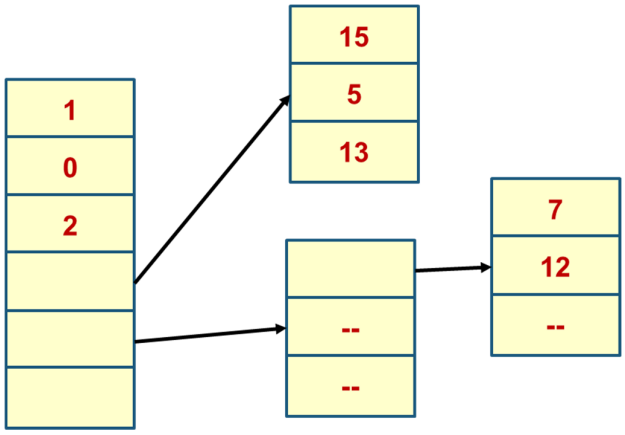
d. **[3 marks]** What do you think is the main difficulty of performing (c) for this disk?

e. **[3 marks]** Suppose the hard disk has only 4 disk blocks (sectors) per track. Can we say the file "**exam.c**" does not incur additional **rotational delay** once we located the first disk block? You can assume we allocate all disk blocks in adjacent track(s).

ANS:

a. 3KB < Size ≤ 4KB (6→9→14→3)

b.



c.

0	1	8	9
1	2	9	11
2	3	10	BAD
3	EOF	11	12
4	5	12	EOF
5	6	13	14
6	7	14	15
7	8	15	EOF

FAT (16 disk blocks)

easy.txt	0
exam.c	4
kidding.bmp	13

Directory Entry (for **root**)

It is important to understand that "bad" code refers to unusable disk sector. So, the "BAD" code should not be moved.

d. Since the disk is full, we need to bring the blocks to memory during the defragmentation.

e. Not really. As "exam.c" occupies 2 tracks, there may be rotational delay between the change of track. Unless this was taken into account during the allocation (i.e. the disk block on the second track is chosen such that the rotational delay is cancelled by seek delay. This idea is known as **disk skew**).

~~~~ End of Paper ~~~~