NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

FINAL ASSESSMENT

AY2022/23 Semester 1

CS2106 – INTRODUCTION TO OPERATING SYSTEMS

DURATION OF ASSESSMENT: 2 HOURS

QUESTION AND ANSWER SHEETS

Instructions (please read carefully):

1. Write down your Student Number on the right and using ink or pencil, shade completely the corresponding bubbles in the grid for each digit or letter. **DO NOT WRITE YOUR NAME!**
2. This answer booklet comprises TWENTY-SIX (26) pages, including this cover page, 30 MCQ questions and 3 short questions.
3. This is an **OPEN BOOK** assessment.
4. You are allowed to bring a calculator, but it cannot have any form of external communication capability. Mobile phones and tablets are not allowed.
5. All questions must be answered in the space provided on this paper; no extra sheets will be accepted as answers.
6. For all MCQ questions, there is only one correct answer. Indicate your answer by shading the bubble to the left of your chosen option.
7. You are allowed to write with pencils, as long as it is legible. Marks may be deducted for unrecognisable handwriting and/or for not shading the MCQ options or the student number properly.
8. Hexadecimal strings are written with the "0x" prefix. All addresses are byte addresses unless otherwise specified.

STUDENT NUMBER

A								
U ○	0	0	0	0	0	0	0	A N
A ●	1	1	1	1	1	1	1	B R
HT ○	2	2	2	2	2	2	2	E U
NT ○	3	3	3	3	3	3	3	H W
	4	4	4	4	4	4	4	J X
	5	5	5	5	5	5	5	L Y
	6	6	6	6	6	6	6	M
	7	7	7	7	7	7	7	
	8	8	8	8	8	8	8	
	9	9	9	9	9	9	9	

For Examiner's Use Only

Question	Marks
MCQ Part	/60
Short Questions	
Q31	/18
Q32	/12
Q33	/10
TOTAL	**/100**

0279

MCQ SECTION

Indicate your answer by shading the bubble next to your answer of choice. For all questions, there is only one correct answer. Each correct answers carry 2 marks.

[General]

QUESTION 1

Which of the following are used by the OS to provide isolation and protection?

i. Virtual memory
ii. Process abstraction
iii. Exception handling
iv. File permissions

ANSWER:

- ○ (i) only.
- ○ (i) and (ii) only
- ○ (i), (ii) and (iii) only.
- ○ None of the above.
- ○ All of the above.

[Process and Process Scheduling]

QUESTION 2

What of the following statement is TRUE about OS processes and threads?

ANSWER:

- ○ Only threads can share memory, not OS processes.
- ○ OS processes can handle signals while threads cannot.
- ○ OS processes can make system calls but threads cannot.
- ○ Each OS process has its own distinct page tables but not threads.
- ○ Threads can use pthread mutexes but OS processes cannot.

QUESTION 3

Why must each thread have its own stack distinct from the other threads?

ANSWER:

- ○ It is needed to support its own procedure calls.
- ○ It is needed for software testing.
- ○ It is so that the thread can use semaphores.
- ○ It will enhance sharing of data.
- ○ It is not necessary.

QUESTION 4

What would be a legitimate use of priority in OS processes?

ANSWER:

- ○ To make background tasks give way to interactive ones so as to improve response time.
- ○ So that a user can get more CPU time relative to others.
- ○ To make C applications run faster.
- ○ Speed up critical section execution so as to release locks earlier.
- ○ None. Priority is just for show.

QUESTION 5

Suppose the following are the actual execution of a process, in between I/Os, given in abstract time units:

Actual Execution $\text{Time}_0 = 2$
Actual Execution $\text{Time}_1 = 13$
Actual Execution $\text{Time}_2 = 9$
Actual Execution $\text{Time}_3 = 19$

Assuming "Predicted Execution Time_0" to be 1, and $\alpha = 0.7$, what is the "Predicted Execution Time_4"?

ANSWER:

- ○ 3.241
- ○ 7.967
- ○ 10.819
- ○ 16.055
- ○ 19.410

[System Calls and Signals]

QUESTION 6

What could be the reasons that the OS makes system calls available to users as mere numbers?

i. It prevents direct access to the OS's address space.
ii. It exposes the least amount of information to the user.
iii. So that malicious software cannot call them.
iv. It prevents other processes from calling the same system calls.

ANSWER:

- ○ (i) and (ii) only.
- ○ (i), (ii) and (iii) only.
- ○ (i), (ii) and (iv) only.
- ○ (ii) and (iii) only.
- ○ None of the above.

QUESTION 7

Why is there a need for a user level library helper to assist with any system call? Choose the most relevant option.

ANSWER:

- ○ It helps check and marshal the necessary parameters for the call.
- ○ It is more memory efficient than direct system calls.
- ○ It is more compute efficient than direct system calls.
- ○ It enables synchronization.
- ○ It is more secure.

[Synchronization]

Questions 8 to 10 below assume the following code. Each question is independent of the other assumes the same initial state that the items 0, 1, 2, 3, 4 (starting with 0, ending with 4) are already pushed successfully onto the stack with **sp** = 5. We shall use the notation '[0, 1, 2, 3, 4]' to denote the status of the stack (leftmost is the bottom, rightmost is the top pointed to by **sp;** **sp** is also the length of this sequence). Also, assume that all unused location of the stack are zeros.

```
int stack[1000];
int sp = 0;

void push(int item)
{
      sp = sp + 1;

      if (sp >= 1000) {
         // ERROR... BOMB!
      }
      stack[sp] = item;
}

int pop()
{
     if (sp == 0) {
        // ERROR... BOMB!
     }
     sp = sp - 1;
     return(stack[sp]);
}
```

QUESTION 8

Suppose there are two threads, one doing "**push(5)**" and another doing "**push(6)**". Which of the following is *not* a possible state of the stack after both threads has completed their push operation?

ANSWER:

- ○ [0, 1, 2, 3, 4]
- ○ [0, 1, 2, 3, 4, 5]
- ○ [0, 1, 2, 3, 4, 6]
- ○ [0, 1, 2, 3, 4, 5, 6]
- ○ [0, 1, 2, 3, 4, 6, 5]

QUESTION 9

Suppose there are two threads, both doing "`pop()`". Which of the following *is* a possible state of the state of the stack after both threads has completed their pop operation starting with the [0, 1, 2, 3, 4] initial configuration?

ANSWER:

- ◯ [0, 0]
- ◯ [0, 1]
- ◯ [0, 1, 2, 3]
- ◯ [0, 1, 2, 4]
- ◯ The stack is empty.

QUESTION 10

Bob learnt about race conditions and decided to put locks into the code. He decided to place line 6 under a lock, i.e., (i) perform a lock operation before executing line 6, (ii) unlock after line 6, (iii) perform a lock again before line 11 and then unlock before executing line 11. Which of the following statements correctly describe what Bob did?

ANSWER:

- ◯ His solution is correct and efficient – coz line 8 is just a read.
- ◯ His solution does not solve the data race.
- ◯ He could have used pthread condition variable.
- ◯ He copied his code from Stack Overflow.
- ◯ He should have used a barrier.

[Memory Management]

Questions 11 to 16 below uses these same set of assumptions. Namely, consider a byte-addressed virtual memory system with the following set up: (i) virtual addresses are 32 bits long, (ii) physical addresses are 28 bits long, (iii) a page is 4Kibyte (4096 bytes), (iv) each page table is a page in length, (v) page table entries are 4 bytes, and (vi) it uses 3 levels of page tables.

QUESTION 11

How many entries are there in a single page table?

ANSWER:

- ○ 256
- ○ 512
- ○ 1024
- ○ 2048
- ○ 4096

QUESTION 12

Suppose a process only has one physical page frame allocated to it. What is the total memory required for the page tables (not including that final one data page frame).

ANSWER:

- ○ 1029 bytes
- ○ 5000 bytes
- ○ 10821 bytes
- ○ 12288 bytes
- ○ 16384 bytes

QUESTION 13

Suppose a process has 10 physical page frames allocated to it. What is the minimal memory space that would be required for the page tables (not including the 10 data page frames)?

ANSWER:

- ○ 10821 bytes
- ○ 12288 bytes
- ○ 16384 bytes
- ○ 36864 bytes
- ○ 45056 bytes

QUESTION 14

Suppose a process has 10 physical page frames allocated to it. What is the *worst case* memory space that would be required for the page tables (not including the 10 data page frames)?

ANSWER:

- ○ 36864 bytes
- ○ 45056 bytes
- ○ 86016 bytes
- ○ 167936 bytes
- ○ 372736 bytes

QUESTION 15

Assuming that processes do not share any pages. In the worst case, how many processes would the system be able to accommodate if each process uses 100 logical pages before it runs out of (maximum) physical memory? Assume that all pages reside in physical memory, i.e., no disk backup.

ANSWER

- ○ 13
- ○ 115
- ○ 217
- ○ 326
- ○ 512

QUESTION 16

Following up on Question 15, i.e., using the same assumptions in that question except that the system turns on demand paging using an infinite disk storage to backup for data pages (but not the page tables – which remains in physical page frames), how many processes can the system accommodate in the worst case? Note that you need at least one data page in physical memory to contain some code and data for execution.

ANSWER:

- ○ Infinite
- ○ 115
- ○ 217
- ○ 326
- ○ 512

Question 17 and 19 is regarding the following memory reference trace:

```
2 3 1 2 3 1 4 2 2 3 1 4 5 2
```

We shall assume: (i) the above are page numbers, (ii) the machine does virtual memory with just 3 physical page frames, and (iii) the page frames are initially empty.

QUESTION 17

How many page faults will there be at the end of the execution in the OPT (optimal) case?

ANSWER:

- ○ 5
- ○ 6
- ○ 7
- ○ 8
- ○ 9

QUESTION 18

How many page faults will there be at the end of the execution if a FIFO eviction algorithm is used?

ANSWER:

- ○ 7
- ○ 8
- ○ 9
- ○ 10
- ○ 11

QUESTION 19

How many page faults will there be at the end of the execution if a LRU eviction algorithm is used?

ANSWER:

- ○ 7
- ○ 8
- ○ 9
- ○ 10
- ○ 11

QUESTION 20

Compared with the standard page table scheme, which of the following statement is FALSE about traditional and inverted page tables? Assume the version introduced in class with no other extensions.

ANSWER:

- ○ Inverted tables are slower.
- ○ Inverted table consumes less memory overhead in general.
- ○ Traditional page tables map logical/virtual pages to physical frames while inverted tables maps physical frames to virtual pages.
- ○ There must be a process ID field in every inverted table entry.
- ○ Both supports sharing of memory pages.

QUESTION 21

Which of the following address falls in the 512-byte size buddy block of this address: **`0x485BB1`**?

ANSWER:

- ○ `0x485BA1`
- ○ `0x4859B1`
- ○ `0x485AB1`
- ○ `0x485CB1`
- ○ `0x486BB1`

QUESTION 22

Following up from Question 21, suppose both the 512-byte block containing **`0x485BB1`** and its buddy are freed. What is the starting address of the resultant 1024-byte block?

ANSWER:

- ○ `0x485800`
- ○ `0x485900`
- ○ `0x485A00`
- ○ `0x485B00`
- ○ `0x486C00`

0279

QUESTION 23

Consider a buddy system allocation system for a memory of size 128 bytes, initially free. The following sequence for memory size are requested (in the following order):

Request 1: 12 bytes
Request 2: 30 bytes
Request 3: 7 bytes
Request 4: Free block assigned for Request 1 (i.e., free 12 bytes),
Request 5: Free block assigned for Request 3 (i.e., free 7 bytes).

After the completion of the above five requests, what are all the free block sizes that are available for future allocation?

ANSWER:

- ○ 64 bytes, 32 bytes
- ○ 64 bytes, 8 bytes, 8 bytes
- ○ 16 bytes, 8 bytes
- ○ 32 bytes, 16 bytes
- ○ None of the above

QUESTION 24

Suppose we have the following 2048-entry inverted table:

PID	Page Number
0	0
1	0
2	0
3	0
0	1
1	1
2	1
3	1
⋮	⋮
0	511
1	511
2	511
3	511

In other words, entry *i* of the table is the entry for PID = *i* mod 4, and page number (*i* >> 2) – "left shift *i* by 2 bits". What is the physical address for Process with ID of 1 and a virtual address of `0xAC529` assuming each page is 4096 bytes?

ANSWER:

○ `0xAC429`

○ `0xACB29`

○ `0x2A0529`

○ `0x2B1529`

○ `0x2AC012`

[File Management]

QUESTION 25

Consider the following code snippet. Assume it executes without compilation or run-time errors.

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<stdlib.h>
int main()
{
    char c1,c2;
    int t;
    int fd1 = open("foo.txt", O_RDONLY, 0);
    int fd2 = open("foo.txt", O_RDONLY, 0);
    do {

        t=read(fd1, &c1, 1);
        t=read(fd2, &c2, 1);
        if (t==0)
            break;
        else
            printf("%c%c ", c1,c2);


    } while(t!=0);
    return 0;
}
```

What is the output? Assume the content of the file foo.txt is **CS2106**

ANSWER:

- ○ `C S 2 1 0 6`
- ○ `CC SS 22 11 00 66`
- ○ `C S 2 1 0 6 C S 2 1 0 6`
- ○ `C C S S 2 2 1 1 0 0 6 6`
- ○ None of the above

0279

QUESTION 26

Consider the following code snippet. Assume it executes without compilation or run-time errors.

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<stdlib.h>
int main()
{
       char c;
       int t;
       int fd1 = open("foo.txt", O_RDONLY, 0);
       fork();

       t=read(fd1, &c, 1);
       while(t!=0){
           printf("%c ", c);
           t=read(fd1, &c, 1);
       }

       wait(NULL);
       return 0;
}
```

How many characters (printed due to %c) will be printed on the screen? Assume the content of the file foo.txt is **CS2106**

ANSWER:

- ○ 10
- ○ 5
- ○ 6
- ○ 11
- ○ None of the above

QUESTION 27

Which of the following statement is FALSE regarding symbolic links?

ANSWER:

- ○ Symbolic links provide a shortcut to access a file it links to.
- ○ Each symbolic link creates a separate I-node entry in the I-node table.
- ○ Deleting the linked file will cause the symbolic link to become 'dangling link'.
- ○ Symbolic links are still valid if the name of the linked file is changed.
- ○ None of the above

QUESTION 28

File system provides abstraction for:

ANSWER:

- ○ RAM
- ○ Hard disk
- ○ CPU
- ○ OS
- ○ None of the above

QUESTION 29

When a file is already opened by process A is opened by process B, which of the following tables will be updated:

ANSWER:

- ○ Per-process file descriptor table
- ○ System-wide open file table
- ○ System-wide V-node table
- ○ All of the above three tables
- ○ None of the above

QUESTION 30

Given the following snapshot of FAT file system with a total of 16 disk blocks, each of size 1KiB (1024 bytes):

0	1	2	3	4	5	6	7
FREE	10	15	FREE	8	13	FREE	BAD

8	9	10	11	12	13	14	15
12	EOF	5	BAD	9	2	FREE	EOF

FAT table

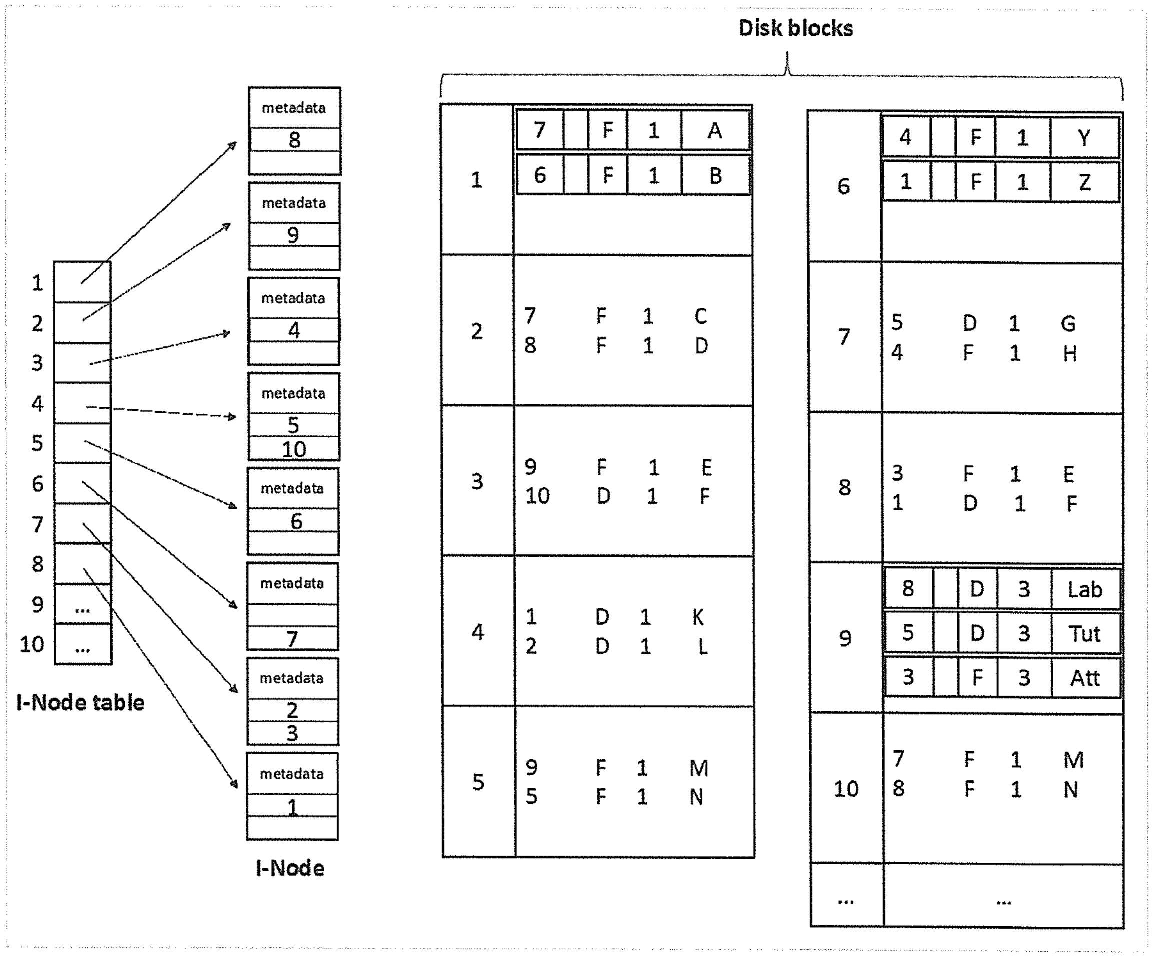
Suppose `file1.txt` is stored in this file system from the ***starting block number of 1***. What is the possible `size` of the file?

ANSWER:

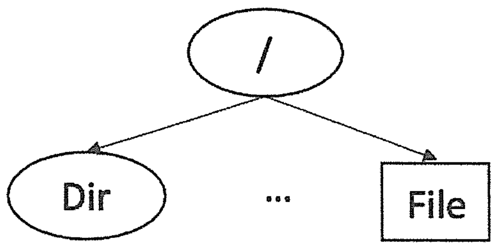
- ○ 4KiB < `Size` <= 5KiB
- ○ 5KiB < `Size` <= 6KiB
- ○ 6KiB < `Size` <= 7KiB
- ○ 7KiB < `Size` <= 8KiB
- ○ None of the above

SHORT QUESTIONS SECTION

QUESTION 31



a) [8 marks] Draw the ***entire directory structure*** from the ext2 file system snapshot. Consider the root directory ("/") to be in I-node number 2. The directory entries in a disk block is as discussed in the lecture *[I-node number, pointer (not shown), Directory (D)/ File (F), filename size, directory/file name].* Follow the tree structure example as shown to represent your directory structure:



0279

Student Number:________________________

Your answer for Question 31(a):

b) [4 marks] Give the ***sequence of I-Node numbers*** (in the correct order of access) to reach the file name "A".

Your answer for Question 31(b):

c) [4 marks] Which ***disk block numbers*** stores the content of the file "Y".

Your answer for Question 31(c):

d) [1 mark] Suppose a new file ("Marks") in created in the root directory ("/"). Which disk block entry will be modified to indicate the creation of the new file in the root directory?

Your answer for Question 31(d):

e) [1 mark] How many new entries will be made in ***the I-node table***?

Your answer for Question 31(e):

QUESTION 32

Use the page number reference string in Question 17-19.

a) [10 marks] Fill in the table below using the CLOCK eviction algorithm.

Your answer for Question 32(a):

Time	Memory Reference	Frame			Fault?
		A	B	C	
1	2				
2	3				
3	1				
4	2				
5	3				
6	1				
7	4				
8	2				
9	2				
10	3				
11	1				
12	4				
13	5				
14	2				

b) [2 marks] How many page faults are there in total at the end of the execution?

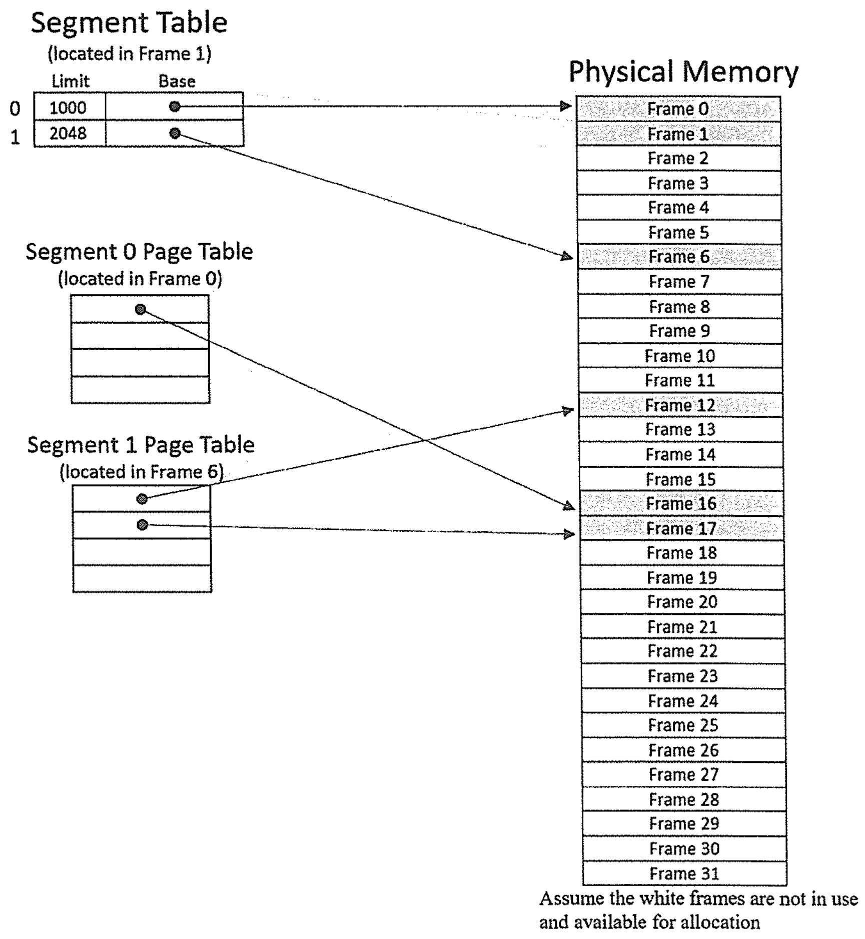
Your answer for Question 32(b):

QUESTION 33

A CPU uses byte-addressed segmented paging with following assumptions (we need to keep things small else this question will become unwieldy):

- An application has only two segments.
- Each segment has its own single level page table.
- Each page table holds 4 entries.
- There are in total 32 physical page frames, each of which holds 1024 bytes.
- The segment limits give the length of the segment in *bytes*.
- The logical address would be 11 bits consisting of a 1 bit segment ID followed by a 10 bit byte offset in the segment, while the physical address consists of a 5 bit frame number followed by the 10 bit byte offset.

This is the current instance of the system:



a) [2 marks] What would be the physical address of logical address `0xF7` (or in binary `00011111110`)?

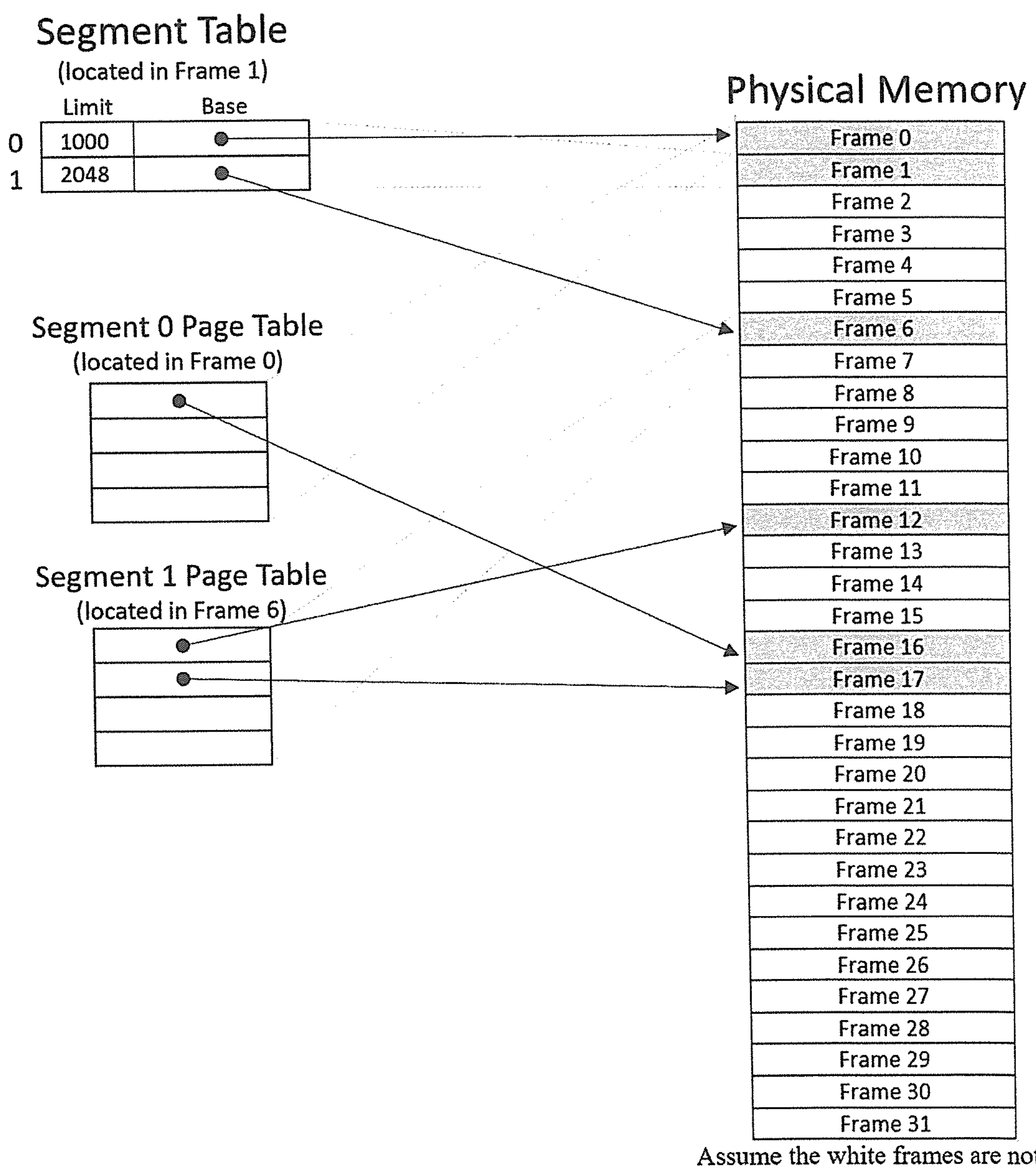
Your answer for Question 33(a):

b) [2 marks] Will this address cause a segmentation fault? Why?

Your answer for Question 33(b):

c) [6 marks] Suppose the application runtime asked for Segment 1 (let's say it's the heap) to be extended by 100 bytes. A system call was made to the OS and the OS allocated Frame 8 to extend the segment. Amend the duplicate figure below to reflect the result after allocation was completed by the application runtime.

Your answer for Question 33(c):



Assume the white frames are not in use and available for allocation

——— *End of Paper* ———