NATIONAL UNIVERSITY OF SINGAPORE

Semester 2 AY2017/18

CS5250 – ADVANCED OPERATING SYSTEMS

Time allowed: 2 hours

DRAFT ANSWERS

WRITE YOUR STUDENT NUMBER IN THESE BOXES

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of **FIVE (5)** questions and comprises **FOURTEEN (14)** printed pages including this page.
2. This is an **OPEN BOOK** assessment.
3. Answer *all* questions. Note that the full mark for each question is different.
4. Write your answers on *this* **QUESTION AND ANSWER SCRIPT**. Answer only in the space (the box) given. Any writing outside this space will *not* be considered.
5. Fill in your Student Number with a pen, clearly on every page of this QUESTION AND ANSWER SCRIPT.
6. You may use pencil to write your answers.
7. At the end of the assessment, please check to ensure that your script has all the pages properly stapled together.
8. Note that when a number is written as "0xNNNN" it means that "NNNN" is in base 16.
9. Approved calculators are allowed for this assessment.

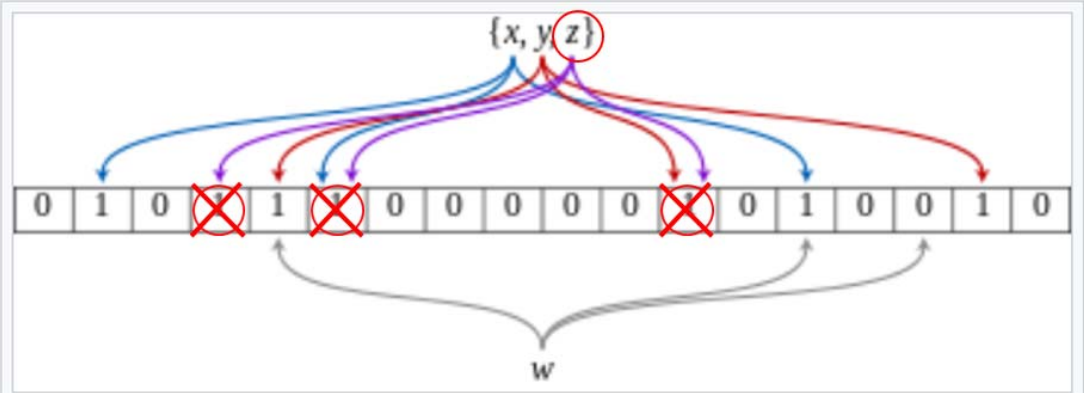
For Examiner's use only

Q1	
Q2	
Q3	
Q4	
Q5	
TOTAL	

QUESTION 1 (15 marks)

(1a) Using an example, show why deletion is not supported in the bit-vector form of the Bloom filter. (5 marks)

ANSWER:



An example of a Bloom filter, representing the set $\{x, y, z\}$. The colored arrows show the positions in the bit array that each set element is mapped to. The element w is not in the set $\{x, y, z\}$, because it hashes to one bit-array position containing 0. For this figure, $m = 18$ and $k = 3$.

There are many possible answers but let's use this one. Deleting **z** will result in the three bit positions turned to zero. Since some of these positions are shared by **x** and **y**, any subsequent query of **x** and **y** will return "not found" – which is incorrect.

(1b) Suppose the single bit for each entry in a Bloom filter is replaced by an (unsigned) byte. Show what modifications to the algorithm you can make so as that deletion is now possible. (10 marks)

ANSWER:

For a counting Bloom filter: on insert, increment the hashed entries using saturating addition – if it hits the largest number – by one, it remains at that number. On deletion, for every hashed entry, decrement by 1 – again using saturating subtraction, i.e., if x = 0, then x – 1 = 0. An object is present if all the hashed locations are non-zero. In this way, deleting an object merely decrement the hash entries. If other objects are hashed to the same entries, then the count would be greater than 1 and hence avoid the situation in (1a).

QUESTION 2 (20 marks)

(2a) Let's assume that we have the following processes entering the system (in the given order):

- P1 burst CPU time: 23, arrival time 0
- P2 burst CPU time: 12, arrival time 5
- P3 burst CPU time: 41, arrival time 10
- P4 burst CPU time: 17, arrival time 15
- P5 burst CPU time: 29, arrival time 40

What is the execution schedule and the completion time for each if we schedule the processes using *pre-emptive shortest remaining time first* with a context switching overhead of 1 time unit? (10 marks)

ANSWER:

1. Time 1-5: P1 runs
2. P2 arrives at time 5, it has shorter remaining time so it gets to pre-empt P1. P1 has 18 units remaining.
3. Time 6: preemption
4. Time 7-18: P2 runs to completion.
5. In the meantime, at time 10, P3 arrives with 41 units and at time 15, P4 arrives at 17 units. However, both are longer than P2, so does not get to preempt P2.
6. Time 19: P4 is the shortest remaining time of 17 units and gets to run after P2 completes.
7. Time 20-36: P4 runs to completion.
8. Time 37: remaining tasks are P1 and P3. Since P1 has 18 units remaining. It gets to run.
9. Time 38-55: P1 runs to completion. In the meantime, P5 arrives but both P3 and P5 require longer time.
10. Time 56: P5 is the shorter of the two and gets to run.
11. Time 57-85: P5 runs to completion.
12. Time 86: remaining task P3 gets to run.
13. Time 87-127: P3 runs to completion.

P1: starts at time 0 and completes at time 55.
P2: starts at time 5 and completes at time 18.
P3: starts at time 10 and completes at time 127.
P4: starts at time 15 and completes at time 36.
P5: starts at time 57 and completes at time 85.

(2b) What is the completion time for each task if we schedule the processes using round robin with a quantum of 10 with no overhead in context switching?

(10 marks)

ANSWER:

1. Time 0-9: P1
2. Time 10-19: P2
3. Time 20-29: P3
4. Time 30-39: P4
5. Time 40-49: P5
6. Time 50-59: P1
7. Time 60-61: P2 - completes
8. Time 62-71: P3
9. Time 72-78: P4 - completes
10. Time 79-88: P5
11. Time 89-91: P1 - completes
12. Time 92-101: P3
13. Time 102-110: P5 - completes
12. Time 111-120: P3
13. Time 121: P3 - completes

P1: starts in time 0 and completes in time 91.
P2: starts in time 5 and completes in time 61.
P3: starts in time 10 and completes in time 121.
P4: starts in time 15 and completes in time 78.
P5: starts in time 40 and completes in time 110.

QUESTION 3 (30 marks)

(3a) Consider the following compare-and-exchange atomic instruction (abstracted as a function – in other words, assume that the following function is atomic):

```
int cas(int *ptr, int oldval, int newval);
```

If the integer pointed to by **`ptr`** is equal to **`oldval`**, then the content pointed to by **`ptr`** will be replaced with **`newval`**, and a '1' will be returned by **`cas`**. Otherwise, a '0' is returned. Using this function/instruction, show how a spinlock can be implemented by giving the C code for the **`spin_lock()`** and **`spin_unlock()`** functions. Write down any assumptions that you make. (15 marks)

ANSWER:

```
// assume this is the default at lock creation
#define UNLOCK 0

// Assume this is long enough for a process ID
typedef int spinlock_t;

int spin_lock(spinlock_t *lock)
{
    int pid = (int)getpid(); // assume PID cannot be 0

    while (!cas((int *)lock, UNLOCK, pid))

    return 1;
}

int spin_unlock(spinlock_t *lock)
{
    int curval = (int) *lock;
    int pid = (int)getpid(); // assume PID cannot be 0

    if (curval != pid) // Not current holder
        return -1;

    *lock = (spinlock_t) 0;
    return 1;
}
```

The lock is 0 if it is unlocked. If locked, it contains the process ID of the lock holder. This is for the unlocking procedure to check if the task attempting the unlock does indeed hold the lock.

(3b) Explain why it would not be straightforward to implement semaphores using basic **`cas`** instructions. What other facilities would you need? (5 marks)

ANSWER:

Semaphores involve blocking which transits a task from RUNNABLE to NON_RUNNABLE. This involves the scheduler API plus recording who is waiting against the semaphore so that a task can be waken up when the semaphore is "up". Hence, it is not merely the use of **`cas`**.

(3c) Suppose we implement the "big reader locks" using the idea outlined in class of per-CPU locks where each reader only needs to get one lock but writers need to acquire all locks. Suppose we have 4 CPUs, i.e., 4 locks. Show a scenario where we have two writers attempting to acquire all locks but resulting in a deadlock. (5 marks)

ANSWER:

Let the two writers be W1 and W2, and the 4 locks be L1, L2, L3, L4. What can happen is that W1 acquires L1, L2, L3, L4 in that order while W2 acquires L4, L3, L2, L1 in that order. Suppose W1 succeeds in acquiring L1 and L2 while W2 succeeds in acquiring L4 and L3. But now, W1 will wait for L3 – which holder, i.e., W2, will wait for L2. This results in a cycle of waits – and a deadlock.

(3d) What is the remedy to fix the problem you identified in **(3c)**? (5 marks)

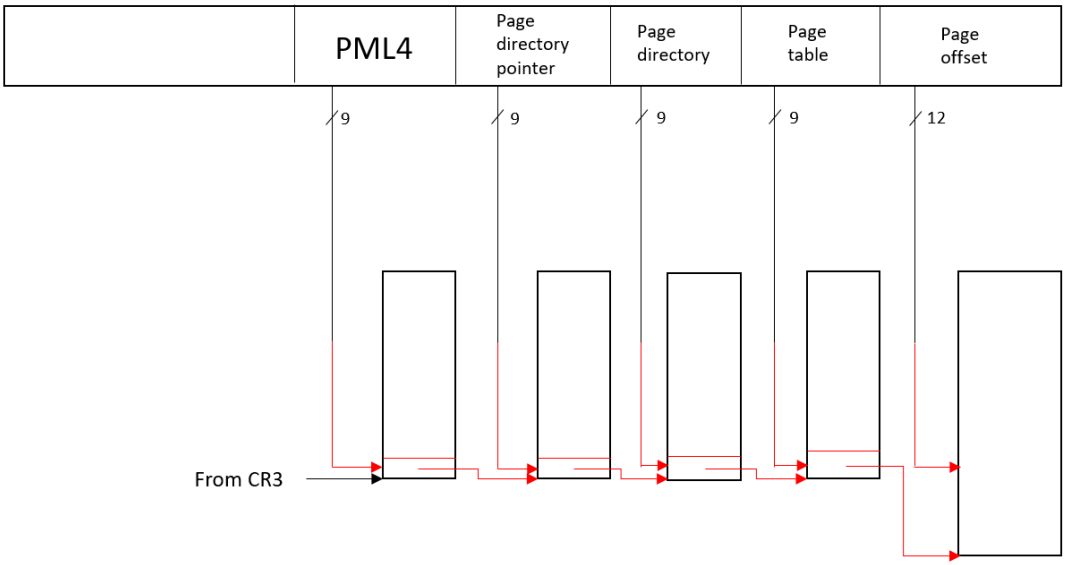
ANSWER:

This is a well known problem. Both W1 and W2 must acquire the locks in the SAME order. If so, only one will succeed in acquiring all locks while the other won't. The order of release (unlock) does not matter.

QUESTION 4 (20 marks)

(4a) Intel's 64 bit architecture uses 9 bits for PML4, page directory pointer, page directory and page table, and 12 bits for page offsets. Suppose a process has only 1 page from address 0 to 4095. Complete the following diagram by completing where the arrows will point to for a virtual address of 0. (5 marks)

ANSWER:



Since the address is all in the page offset bits, all the other bits are zero. So it will be the zero'th entry and the zero'th table every where else.

(4b) How much total memory would be needed for (4a)? Include the last one page that contains data. (5 marks)

ANSWER:

Simply, $(2^9 \times 4 \times 8) + 2^{12} = 20{,}480$ bytes.

(4c) Recalculate the memory required if all the 9-bit quantities in (4a) is now 10 bits. The process still only has a single page of data. (5 marks)

ANSWER:

$(2^{10} \times 4 \times 8) + 2^{12} = 36{,}864$ bytes.

(4d) What would be the advantages and disadvantages of going from 9 bits to 10 bits as done in (4c)? (5 marks)

ANSWER:

Advantages:

1. It increased the total physical address space from 48 to 52, making the addressable physical memory limit to go up from 2^{48} = 256TB to 2^{52} = 1024TB.

2. The same four tables (whose size increased from 16K bytes to 32K bytes) can accommodate 4,096 pages instead of 2,048 pages.

Disadvantages:

1. For a process that uses only a very small number of pages, there is a significant increase in overhead.

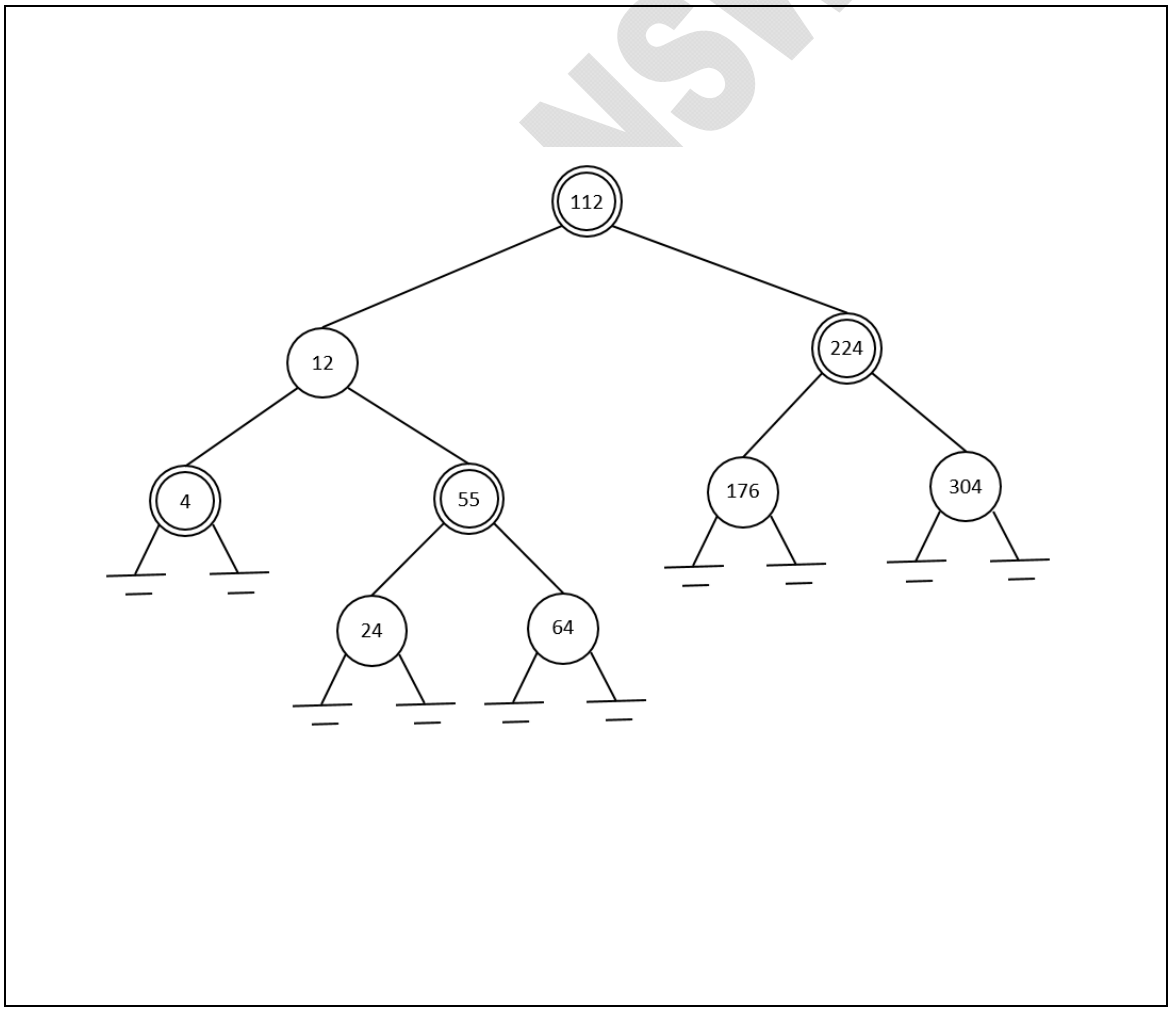
QUESTION 5 (15 marks)

(5a) Draw the final red-black tree after the following insertions are completed. Use a single circle to represent a "red" node and a double concentric circle to represent a "black" node. If you use other notations, make sure you give a legend.

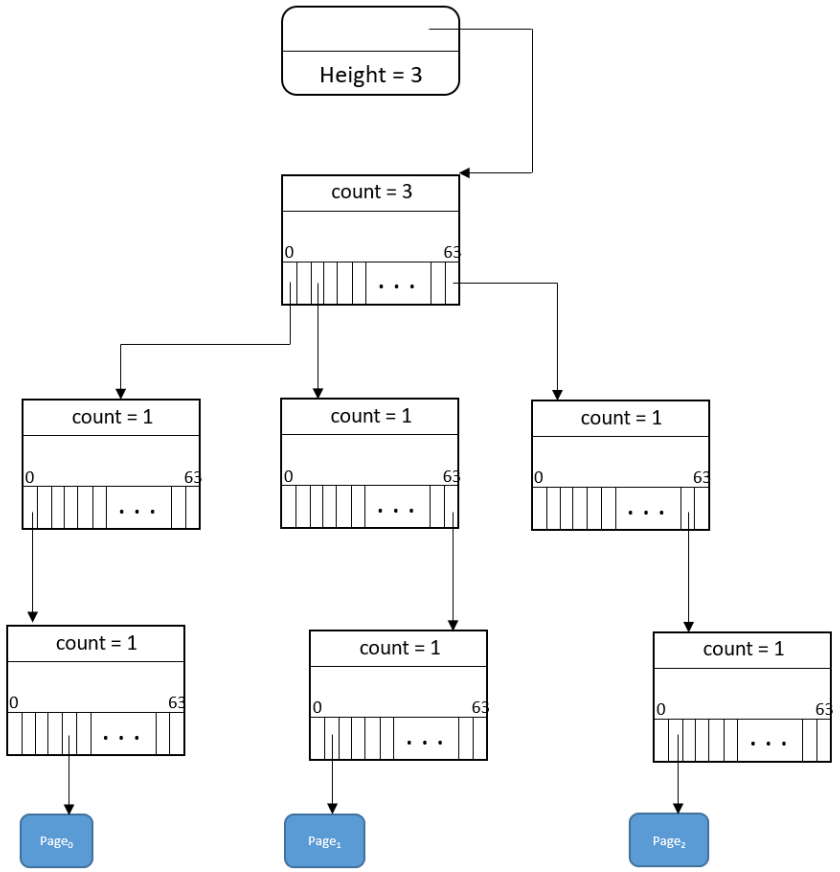
- Insert 112
- Insert 12
- Insert 304
- Insert 4
- Insert 55
- Insert 176
- Insert 64
- Insert 224
- Insert 24

(10 marks)

ANSWER:



(5b) Given the following Linux radix tree, what are the indexes of the three pages (in base 10)?



(5 marks)

Index of Page$_0$ in the diagram = 4_{10}

Index of Page$_1$ in the diagram = $12{,}225_{10}$

Index of Page$_2$ in the diagram = $262{,}017_{10}$

=== END OF PAPER ===