COL331/COL633 Major Exam
Operating Systems
Sem II, 2016-17

Answer all 14 questions (12 pages) Max. Marks: 41

For **True/False** questions, please provide a brief justification (1-2 sentences) for your answer. No marks will be awarded for no/incorrect justification. We will penalize you if you provide extraneous information not related to the question being asked.

1. True/False: On x86 architecture, the TLB needs to be flushed on every context switch [2]

2. The page size in our current systems is 4KB. Give an example of a workload which will greatly benefit if the page size was doubled (8KB). The workload need not be anything useful, and may just be an access pattern in the virtual address space. [3]

3. The page size in our current systems is 4KB. Give an example of a workload which will greatly benefit if the page size was halved (2KB). The workload need not be anything useful, and may just be an access pattern in the virtual address space. [3]

4. Give one advantage and one disadvantage of using large (4MB) pages over regular (4KB) pages. [2]

5. How can an OS decide whether to use large pages or regular pages for a region of virtual memory? Give one concrete example where large pages are typically used by the OS. [3]

6. Give an example of a workload that will perform better with “Random” cache replacement policy than with the CLOCK algorithm to replace the page-cache. (A page cache is the cache for the virtual memory pages in physical memory). [3]

7. In a particular system and workload (with virtual memory and swapping enabled), if four processes A, B, C, D are running together context-switching at 10ms granularity, the system runs very very slow. However if two of the processes (A and B) run for 10 seconds, and then two other processes (C and D) run for the next 10 seconds (and this is repeated every 20 seconds), the system runs *much* faster (i.e., all four jobs finish much quicker). What is the likely reason? Roughly, how much faster can the second scheme (running two processes for 10 seconds at a time) be compared to the first scheme? [3]

8. Assume you want to implement blocking locks, and you have the following abstractions available to you:
    a. spinlock_t with acquire() and release() functions with standard locking semantics (but with spinning)
    b. sleep() and wakeup() with the standard condition variable semantics (as in xv6).

Implement a blocking lock, i.e., implement a data-type blocking_lock_t, and two functions blocking_acquire() and blocking_release() using the spinlock and sleep/wakeup abstractions. [4]

9. Consider the following function ‘remove_node’ to remove a key from a singly-linked list. It assumes that the key is not present in the head (first node) of the linked list. It returns the removed node if found. It returns NULL if the key is not found.

```
struct node {
    int key;
    struct node *next;
};

node *
remove_node(node *head, int key)    //assume that the head node does not contain ‘key’
{
    while (head) {
        if (head->next != NULL && head->next->key == key) {
            node *ret = head->next;
            head->next = head->next->next;
            return ret;
        }
    }
    return NULL;
}
```

a. If multiple threads execute this code in parallel, what are some bad things that can happen? Assume that it has been ensured that the concurrent threads will try and search/remove the same key. Also assume that the keys in the list are unique. Give one example of an undesirable situation. [2]

b. How can you fix the problem using coarse-grained locks, i.e., one lock for the entire list? Write the code that you will need to change/add. [1]

c. How can you fix the problem using the compare-and-swap instruction cmpxchg(), as discussed in class? Write the code that you will need to change/add. [3]

10. Sleep/wakeup

a. Almost all calls to the sleep() function in xv6 are inside a loop, e.g.,

```
while (...) {
    sleep(..., ….);
}
```

Why? [1]

b. Look at line 2683 in xv6 code

2683 : if(!havekids || proc−>killed){

If this condition is true, the function returns, else it goes to sleep. What is the significance of checking (!havekids)? What is the significance of checking (proc->killed)? If the check on (proc->killed) was removed, would the code remain correct? What are some bad things that may happen? [3]

11. When not using logging, and using ordering of writes to disk, to ensure crash recovery, what are the types of inconsistencies that may still happen? Give some examples of what inconsistencies can still happen, and what inconsistencies are prevented? [2]

12. True/False. When not using logging (and using ordering as in the question above), the order in which locks are acquired on blocks and the order in which they must be flushed to disk, is always identical. If True, explain why. If false, give counter-example. [3]

13. In logging-based crash recovery, explain the disadvantages if the transaction size is too small (e.g., one transaction per filesystem operation)? Also, explain the disadvantages if the transaction size is too big (e.g., one transaction very two hours)? [2]

14. After having done the OS course, what was your favourite OS topic? [1]
    a. OS abstractions (e.g., system call interface design)
    b. Instruction Set Architecture for implementing OS abstractions
    c. Traps/Interrupts and privilege separation by architecture
    d. Virtual memory (segmentation, paging and swapping)
    e. Scheduling
    f. Concurrency control (locking, condition variables, etc.)
    g. Device drivers
    h. Filesystem design
    i. Crash recovery
    j. None of the above. In this case, mention your favourite topic (if any).