CSL373/CSL633 Major Exam
Operating Systems
Sem II, 2012-13
May 6, 2013

Answer all 8 questions Max. Marks: 56

1. True or False. Give reasons and/or brief explanation. No marks for incomplete/wrong explanation.

a. On xv6, the walkpgdir() function gets executed on every memory access by the user process. [1 mark]

b. For most desktop applications, using huge pages (e.g., 2GB pages) will result in better overall system performance. [1 mark]

c. An in-memory filesystem, RAMFS, as discussed in the xsyncfs paper, will always have better performance than any on-disk filesystem for a disk-intensive benchmark. [1 mark]

d. In a ext3 journaling file system, a transaction can be closed at any time. [2.5 marks]

e. In xv6, a file create can fail even if the disk is not full. [2 marks]

f. On a FAT32 filesystem, corruption of one disk block could potentially lead to the loss of complete file data. [2 marks]

g. The best possible filesystem layout for a read-only filesystem (which is written only once in the beginning) is contiguous allocation. Explain why or why not. [2.5 marks]

h. When using a write-through buffer cache, one does not need to worry about the order of disk writes (from a filesystem consistency perspective across power reboots). Explain why or why not. [2 marks]

i. The “First Fit” algorithm is used when allocating pages to processes. [1 mark]

2. In xv6, the kernel stack (kstack) of a process is limited by KSTACKSIZE (4096 bytes). List at least three invariants that the kernel maintains to ensure that kstack never overflows. i.e., if one of those invariants is violated, kstack could potentially overflow. [5 marks]

3. The L-1 cache of a processor could either be physically indexed (i.e., indexed using physical addresses), or virtually indexed (i.e., indexed using virtual addresses). List the primary advantage of using a virtually addressed L1 cache. List the primary advantage of a physically addressed L1 cache. [3 marks]

4. Professor X really liked the xsyncfs paper, but found it a bit “incomplete”. He suggests that external synchrony should be implemented over a cluster of machines, to achieve its real potential. He is developing a new filesystem called “networked externally synchronous file system”, or *nxsyncfs*. In nxsyncfs, an output is not considered *external*, until it leaves to the external network. In other words, all communication between two hosts within the cluster is considered *internal*. Assuming that a cluster of machines can only be observed through the external network interface (assume there is no console access to these machines), answer the following questions:

i. What will be your filesystem sync policy on nxsyncfs? i.e., when will you insert fsync calls? Does nxsyncfs give more optimization opportunities over xsyncfs? Briefly explain. [2 marks]

ii. What kind of data structures and mechanisms will you need to implement for nxsyncfs? Assume that all xsyncfs mechanisms are already in place. Only emphasize the extra structures and mechanisms that you will need to add over xsyncfs, to take advantage of nxsyncfs semantics? [3 marks]

iii. Give an example of an application that will show significant improvements with nxsyncfs over xsyncfs. i.e., the performance of the application should significantly improve if nxsyncfs is used in place of xsyncfs. [3 marks]

5. Producer-consumer with semaphores

Consider the following functions, produce(item), and consume() for a shared queue, shared between multiple producers and multiple consumers. As discussed in class, semaphores are used to synchronize access to this shared queue.

```
void produce (item):
        sema_down(mutex);
        sema_down(holes);
        insert_item(item, buffer);
        sema_up(items);
        sema_up(mutex);


item consume(void):
        sema_down(mutex);
        sema_down(items);
        item = remove_item(buffer);
        sema_up(holes);
        sema_up(mutex);
```

In this code, the semaphore “mutex” is used for mutual exclusion, semaphore “holes” is used to count the number of empty slots in the buffer, and semaphore “items” is used to count the number of items in the buffer.

a. What should be the initial values of the semaphores mutex, holes, and items? Assume that the buffer size is N. [1.5 marks]

b. Look at this code closely and answer if this code is correct. If you think it is correct, briefly explain the invariants that prove it correct. If you think it is incorrect, explain why and fix it so that it becomes correct (with brief explanation as before). [4.5 marks]

6. Consider the ext3 journaling filesystem. Answer the following questions:

a. How does ext3 ensure a low runtime overhead, even though a disk write results in at least two separate disk writes (one to log, and one to FS tree). [1.5 marks]

b. Consider a situation where filesystem compound transaction X has started committing. What happens to the transactions (or atomic operations) that are already ongoing? Do the updates of those operations get reflected in the next compound transaction (say transaction Y), or do they get reflected in the previous transaction (transaction X)? [2 marks]

c. What happens to the operations that were started after transaction X started committing but before the commit record was written? Do the updates by these operations get reflected in the previous transaction (transaction X) or the next transaction (transaction Y)? [2 marks]

d. While transaction X is committing (i.e., commit record has not yet been written), what happens if some operation in transaction Y writes to a disk block which is also a part of transaction X? How does ext3 ensure correct behaviour? [3.5 marks]

7. Implement reader-writer locks using spinlocks and sleep/wakeup. You must not use any other synchronization mechanism. You should also not assume specific architecture instructions like xchg, etc. Your implementation should be efficient.

No scheduling requirements: The scheduling of read/write locks can be arbitrary, i.e., if a reader (potentially many) and a writer are both waiting for a lock, any one can win. Also, if a lock is being held in read mode, other readers may be allowed to acquire it, even if a writer is waiting.

You have to implement: struct rwlock, read_acquire(), read_release(), write_acquire(), write_release() functions.

You are allowed to use: spinlock_acquire(), spinlock_release(), sleep(), wakeup(). [4 marks]

8. Security: Explain how submit-pintos script is able to both write to my (sbansal's) home as well as to your (the person who is submitting) home. [3 marks]

9. How does an OS detect Thrashing? [1.5 marks]. How does it deal with it? [1.5 marks]