CSL373/CSL633 Major Exam
Operating Systems
Sem II, 2014-15

Answer all 11 questions (14 pages) Max. Marks: 62

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. If the disk is fully utilized (i.e., providing near-full disk bandwidth), does that imply that the scheduler is doing a good job interleaving CPU-bound and I/O-bound jobs? [4]

Not necessarily. This does not tell us anything about response time e.g. short accesses vs long accesses. It also does not tell us anything about fairness.

Alternative:
No, we can come up with a biased scheduler which prefers I/O-bound jobs, it will have high disk utilization and interleaving will be poor between CPU-bound and I/O-bound jobs.

0 to 1.5 Marks for partially correct
4 for correct

2. Persistent data can be stored either on the local disk of a machine or on the disk of a remote machine (e.g., NFS). In one such system which stores persistent data on remote disk, it also caches the file data in local memory, so that it does not need to go to the remote disk on each read access.

a. Should the filesystem block size be smaller or larger than the block-size of a local filesystem? (Possible answers: smaller, larger, same). Briefly explain. [3]

Assuming fast network, the most performance constrained resource is still the remote disk and local memory. The only thing that has changed is that we have a larger cache (local + remote). But the block size is largely uncorrelated with cache size. So correct answer is: same.

3 marks for correct explanation and 0-1.5 for partial answer

b. The filesystem engineer varies the filesystem block size from 512B to 4MB, and notices that her application performance first increases (with increasing block size) and then decreases (with increasing block size). The application primarily performs reads on the filesystem. Can you explain this behaviour (both increase and decrease)? What do you think is the likely inflection point and why? (Hint: inflection point is the block size at which the application performance was highest). Try and justify your answer by picking a realistic example of the application and the system configuration. [6]

Increasing block size:
Pros:

- **Better exploitation of spatial locality**
- **Better disk bandwidth utilization**

Cons:

- **Cache pollution potentially**
- **Too much network/disk bandwidth usage**

Till a limit increasing block size provides advantage. After that, it hurts. A good value would be application specific.

Example:

- **Web server streaming HTML files ~ 4-16 kB**
- **Video streaming server ~ 512kB - 2MB**

With a typical system config of modern hardware as discussed in calss.

Marks: 4(explanation) + 2(example)

c. Should the filesystem engineer implement a write-back cache or a write-through cache? Can you think of a workload where write-through cache would perform better than a write-back cache? Clearly state your assumptions and answer in 1-2 sentences. [4]

WB cache seems most natural. However WB cache requires consistency management for multiple NFS clients.
Usually WB would perform better. WT could perform better if there are a lot of evictions but even though the better performance would only be transient.

d. In your opinion, what should be a good crash-consistency semantics for this remote filesystem? Justify your answer by considering tradeoffs between performance and crash-consistency semantics. What support would you provide for the application such that it can make assumptions against crash consistency? After stating your solution, clearly enumerate the different scenarios (e.g., what crashes, when does it crash, how recovery happens). [6]

For performance:

- **sync at periodic interval (e.g. 30 sec)**
- **application should be able to force sync (e.g. fsync)**
- **Metadata consistency should be ensured using either ordered writes or journaling/logging.**

Scenarios:
Data:

- **Crash before app calls fsync: all data lost**
- **Crash after fsync has returned: data persistent**

Metadata:

- **Crash before commit: transaction lost**
- **Crash after commit: FS consistent**

Marks: 3+3

3. What is the problem if you run a database server as a user-process over a UNIX-like kernel (e.g., Linux)? How can an exokernel design solve this problem? Your answer should be clear and precise. [2 + 4].

Problems:

- Double eviction (os and app)
- Potentially two schedulers for CPU and disk

Exokernel provides hardware like view. All policy can be implemented inside the application, specific to it. It will prevent double eviction (explain).

4. In which of these situations can an intruder be able to compromise the security of the system. Assume that sensitive (private) information lives in the registers, memory, and the disk of the system. A security compromise means that the intruder can gain access to this sensitive information.

a. Intruder has access to the raw disk device storing the passwords [1]

Easy. A simple scan of disk contents (bypassing OS) will expose the password.

b. Intruder has access to the power supply of a system and has a user (non-root) account on the machine. Access to power supply means that she can turn the machine on/off (through power supply) at any time, and any number of times. Assume that the filesystem implements an asynchronous crash consistency model, whereby the filesystem data is flushed to disk only at periodic intervals. You can also assume a specific filesystem (e.g., ext3) while answering this question. You may also define your own simple filesystem to answer this question. Your answer may depend on the choice of filesystem. Notice that the intruder may have a user account, but could compromise security by getting access to sensitive data of another user/root user. [6]

If using ext3, no attack possible. The power failure will do nothing but to keep the FS in consistent state.
(expect some more explanation)

5. True/False: An ext3 transaction can be made to contain an arbitrarily large number of operations? If true, explain why. If false, name at least three different criteria on which the maximum size of an ext3 transaction would depend [4]

False. (0.5 marks)

Max size depends upon:

1. **Disk size (for correctness)**
2. **Memory size (for performance)**
3. **Crash consistency guarantees (for user experience)**

(1- 2 marks, 2 & 3- 1.5 marks)

6. Why do we need separate “close” and “commit” operations for an ext3 log? Briefly explain in 1-3 sentences. Incomplete answers will not receive marks. [4]

To gracefully allow ongoing operations to finish before committing the current transactions.
All the new operations belong to the next transaction (after close).

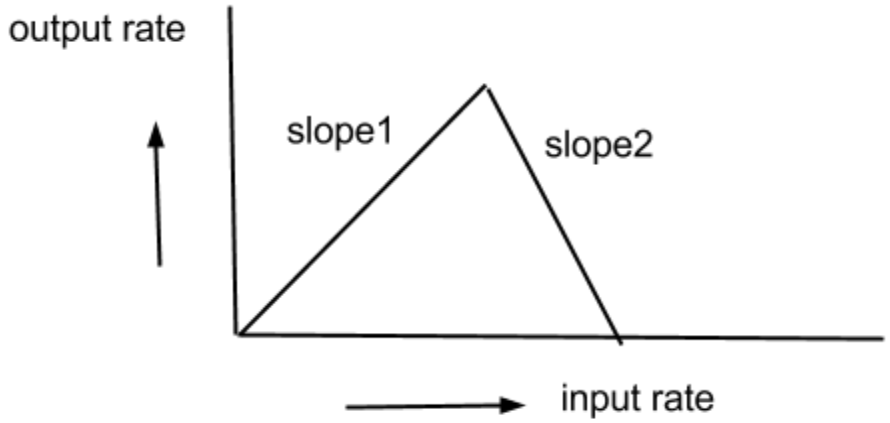
7. Explain in one sentence, why RCU is better than reader-writer locks? [2]

Allows reader to execute at cache speed
Does not introduce cacheline bouncing.

8. Explain in 1-2 sentences why transactional memory (or transactions in general) are often considered better than locks? [2]

No deadlock.
No loss of modularity.
No headache associated with fine grain locking.

9. During receive livelock, the system may behave as follows (as also discussed during lecture):



What determines slope of line 1 (slope1)? What determines slope of line 2 (slope2)? [5]

Slope 1: (1.5 marks)
45 degree or 1 always

Slope 2: (3.5 marks)
Depends on the time it takes to handle interrupt. Higher interrupt handling time w.r.t. output processing time => steeper slope

10. The following code attempts to implement an atomic stack. Does it work? If yes, give a short intuitive correctness argument why it works. If no, state how it is vulnerable to race conditions. Assume the stack is initialized correctly. [6]

```
struct stack {
    struct lock head_lock;   // used to lock head
    struct elem *head;       // stack top
    lock_t count_lock;       // used to lock count
    int count;
};
struct elem {
    struct elem *next;
     /* other data fields. */
};
void push(struct stack *s, struct elem *e) {
    acquire(&s->head_lock);
    e->next = s->head;
    s->head = e;
    release(&s->head_lock);
// V operation
    acquire(&s->count_lock);
    s->count++;
    release(&s->count_lock);
}
struct elem *pop(struct stack *s) {
    struct elem *e;
    int acquired;
// P operation
    for (acquired = 0; !acquired; ) {
        acquire(&s->count_lock);
        if (s->count) {
            s->count--;
            acquired = 1;
        }
        release(&s->count_lock);
    }
    acquire(&s->head_lock);
    e = s->head;
    s->head = e->next;
    release(&s->head_lock);
    return e;
}
```

There is no race condition.

The count acts as semaphore. Look at the comments in the code on last page.
Even if interleaving takes place after the P/V operation the count will be consistent.

Marks:
0 if no explanation or wrong answer
6 for correct answer and explanation.

11. True / False : Increasing the size of the buffer cache should always reduce the (average) number of disk accesses. Assume everything else (e.g., workload, hardware, OS logic, etc.) is kept same, only buffer cache size is increased. [3]

True

False (is also valid):

- **Belady's anomaly**
- **Cache replacement policy can be arbitrary**
- **Increasing buffer cache decreases swap cache, so VM disk writes to swap space can increase.**

True (with no explanation) full marks
False (with any of the correct explanation) full marks
False (with no explanation) 0 marks