CS372H: Spring 2008 – Final Exam

Instructions

- This final is closed book and notes.
- If a question is unclear, write down the point you find ambiguous, make a reasonable interpretation, write down that interpretation, and proceed.
- State your assumptions and show your work. **Write brief, precise, and legible answers.** Rambling brain-dumps are unlikely to be effective. **Think before you start writing** so that you can crisply describe a simple approach rather than muddle your way through a complex description that "works around" each issue as you come to it. **Perhaps jot down an outline** to organize your thoughts. And remember, a **picture can be worth 1000 words.**
- For full credit, show your work and explain your reasoning.
- There are 11 problems on this exam. The first 10 are worth 8 points each and the last is worth 20.
- The exam is scheduled Monday May 12 2-5PM JGP 2.102. The exam is not designed to require the full 3 hours to complete.
- Write your name on this exam

1. (8) In the context of authentication protocols, define the meaning of "jurisdiction" and give an example of its use.

   Solution: If node A trusts node B's statements on some subject, then A believes B has jurisdiction. More formally, if A believes B has jurisdiction for statement X and A beleives B believes X, then A believes X.
   Example: if A trusts B to be a key authority for C, and A believes B believes K is a good key for A and C to use to communicate, then A believes K is a good key to use for communication with C.

2. (8) What protocol can be used to ensure that two nodes are guaranteed to take the same action at the same time?

   Solution: No such protocol exists. It is impossible to guarantee this behavior. 2 generals problem

3. (8) Why is a bitmap a better choice than a linked list for tracking free disk blocks?

   Solution: Need to be able to place related data items near one another on disk. Bitmap can be indexed by location, while linked list cannot (at reasonable cost).

4. (8) In lab 4, the `faultalloc` testprogram allocated a new page at `va` on a page fault for a bad reference to `va`. The `faultalloc` program's `umain` looks like this:

```
void
umain(void)
{
set_pgfault_handler(handler);
cprintf("%s\n", (char*)0xDeadBeef);
cprintf("%s\n", (char*)0xCafeBffe);
}
```

   You also ran `faultallocbad`, which replaced the `cprintf`s above with a call `sys_cputs((char *)0xDEADBEEF, 4)`. If things were working properly, `faultallocbad` died rather than allocate a new page. The exercise asked you to understand why `faultalloc` and `faultallocbad` behaved differently.

   Why do `faultalloc` and `faultallocbad` behave differently?

   Solution: In faultalloc, the cprintf library function is users space code that attempts to copy data from 0xDeadBeef and 0xCafeBffe to a buffer. This user space code references illegal memory, causing a page fault that is sent back to user space and fixed. IN contrast in faultallocbad, we do a system call that causes the kernel to try to access bad memory, so the kernel fault handler just shuts downt he user program.

5. (8) In file systems, what is a *hard link*?

   Solution: Hard links allow multiple directory entries to refer to the same file number.

6. (8) Why is it difficult to implement hard links in the FAT file system?

   Solution: Hard links allow multiple directory entries to refere to the same file number. But since FAT has no inode it stores file metadata in directory entry. If we have multiple entries, which should store metadat (e.g., owner, modified time, permissions, link count); how to find other copies when one is changed? Worst: how to store reference count so that delete/unlink works.

7. (8) An important operating system architecture is a *virtual machine monitor* or *hypervisor*. A hypervisor can run multiple unmodified operating systems on separate virtual machines—each guest operating system believes it is running on its own machine, and the guest operating systems are isolated from one another so that a bug in one cannot cause a problem in another.

   One of the challenges to making this work is that guest operating systems must run with the USER status bit set so that the hypervisor can limit what guest OSes can do. As a result, guest OS code cannot execute privileged instructions like `lcr3()` to install a page table for itself (e.g., the kernel page table for the guest OS) or for processes that the guest OS creates.

   Describe how a hypervisor could ensure that unmodified guest operating systems can function properly (while still ensuring isolation among the virtual machines and while still ensuring that processes within the guest OS are isolated from one another.)

   Solution: Two key ideas. (1) hypervisor emulates privleged instructions for each virtual machine: when a guest OS executes a privleged instruction, hardware issues a trap and calls hypervisor handler which can simulate the instruction for the guest OS; (2) need to track whether virtual machine is operating in virtual user mode or virtual kernel mode so that if user process tries to execute privleged instruction we call guest OS handler but if guest OS executes privleged instruction we emulate it. Track virtual user/supervisor mode by tracking trap/interrupt/system call/iret instructions, all of which are dispatched to hypervisor handlers before being bounced back down to guest OS handler.

8. (8) Consider the following sequence of reads and writes (assume all values are initially 0; each operation occurs at the exact real-time moment specified for each line):

Time	Node 1	Node 2	Node 3	Node 4
0:01	write(A, 12)	write(A, 22)		
0:02	write(B, 13)	write(B, 23)		
0:03	write(C, 14)	write(C, 24)	23 = read(B)	13 = read(B)
0:04	write(D, 15)	write(D, 25)	12 = read(A)	12 = read(A)
0:05			25 = read(D)	25 = read(D)
0:06			13 = read(B)	13 = read(B)

Indicate whether each of the following statements is true or false and explain your answer.

(a) The above system implements linearizability

Solution: false. At time 3 we have two different results for write of B at time 2.

(b) The above system implements sequential consistency

Solution: True. An order consistent with all operations listed above is write(A,2), w(A,12), w(B,23), w(B,13), w(C,14), w(C, 24), w(D,15), w(D,25)

(c) The above system implements causal consistency **Solution:** true

9. (8) Although passwords have limitations, sometimes they must be used for authentication. If you must design a system that uses passwords for authentication, describe three ways to improve their security.

Solution: force users to use long/varied passwords; don't store passwords (store hash of password + salt); limit rate of password guessing by adding delay after missed password; don't send cleartext password across network (send challenge/response instead)

10. (8) The MegaGiga hard disk rotates at 10000 rpm (6ms/rot)with a seek time given by = 1 + 0.001t msec, where t is the number of tracks the arm seeks. Assume a block size of 512 bytes, 1024 sector/track, 8192 tracks, and 4 platters. The disk has a 16MB track buffer. The disk controller can DMA read or write data between memory and the disk device at a rate of 100MB/sec.

    (a) What is the storage capacity of this disk?

    Solution: 2^9 bytes/sector * 2^{10} sectors/track * 2^{13} tracks/platter * 2^2 platters/disk = 2^{34} bytes/disk = 16GB

    (b) Estimate the worst case delay to read 512 bytes from this disk

    Solution: worst case seek is 1+.001*8192 = 1+8.192ms = 9.192ms
    worst case rotation time is 15000 rev/minute * 1/60 minute/second = 6ms
    total = 15.2ms

    (c) Estimate the expected time to read 20 consecutive MB from a random location on disk.

    Solution: Each track holds .5MB, so we need to read 39 full tracks + part of a first track and part of a last track (only a 1 in 1024 chance that a request is exactly aligned with the start of a a track, so we can ignore that case as having little impact on expected performance. We will have one random seek and 40 1-track seeks (or head switches) → **total seek time** = 40ms + 1 + 8192/3 * .001 = 40 + 1 + 2.7 = **43.7ms**
    for 39 tracks, we sit on the track for a full rotation. For the last, we will assume that tracks are offset enough to compensate for the seek, so we have to wait for 1/2 rotation on average (if tracks are badly laid out, then we could always have to wait for a full rotation.) Finally, for the first track, if we have to read p percent of it, then there is a p chance we'll have to wait for a full rotation (since we arrive "in the middle" of the track); there is a 1-p chance we'll land in the middle of the free part and have to wait (on average) (1-p)/2 + p = 0.5+.5p, so we have integrate(p = 0..1)(1.5 + .5p) = $.375p^2$ = .375. So we have **total rotation time** = 40.375 rotations * 6ms/rot = **242.2ms**
    This gives us a **total time** of **285.9ms** for 20MB, which is under 80MB/s, so the bus bandwidth doesn't limit us.

11. (20) Concurrent programming

Consider the following, simple game: three player each have an unlimited supply of color cards. Player Blue has a big stack of blue cards. Player Green has a big stack of green cards. Player Red has a big stack of red cards. The Dealer (a fourth entity) has a big stack of blue cards, a big stack of green cards, and a big stack of red cards.

The Dealer randomly selects two different colored cards and places them on the table. The player who has the third color removes the two dealer cards from the table and takes the three cards (two from the table and one from the player's hand) and places them in a pile. The Dealer then puts another two cards on the table, and the cycle repeats.

Write a `Table` object to synchronize game play among four threads (one thread each for Dealer, Red, Blue, and Green). The table object has two public methods:

```
public class Table{
 public:
   int const BLUE = 0;
   int const GREEN = 1;
   int const RED = 2;

   dealerPutCards(boolean cardList[3]);
   playerPullCards(int playerCardColor);

 ...
}
```

The method `dealerPutCards()` passes an array of three booleans, two of which are true and one of which is false. It returns once the matching player has played.

The method `playerPullCard()` passes in an `int` representing a color. It returns when the player's color is the winning color.

To receive credit on this problem, you must follow the coding standards described in the handout and project.

List Table's member variables and indicate how they are initialized

Solution:

Lock mutex = new Lock();
Condition dealerDone = new Condition();
Condition playDone = new Condition();
boolean cardsOnTable[] = {0,0,0};

Implement dealerPutCards(boolean cardList[3])

Solution:

```
mutex.lock();
memcpy(cardsOnTable, cardList, 3 * sizeof(boolean));
    dealerDone.signal();
    while(cardsOnTable[BLUE] || cardsOnTable[GREEN] || cardsOnTable[RED])
    playDone.wait(&mutex);
        mutex.unlock();
```

Implement playerPullCards(int playerCardColor)

Solution:

```
mutex.lock();
while(notMyWin(), playerCardColor)
    dealerDone.wait(&mutex);
        cardsOnTable[BLUE] = cardsOnTable[GREEN] = cardsOnTable[RED] = false;
    playDone.signal();
    mutex.unlock();
```