Stanford University
Computer Science Department
CS 240 Quiz 1 Spring 2009

April 26, 2010

!!!!!! SKIP 20 POINTS WORTH OF QUESTIONS. !!!!!

This is an open-book exam. You have 75 minutes. Cross out the questions you skip. Write all of your answers directly on the paper. Make your answers as concise as possible. Sentence fragments ok.

NOTE: We will take off points if a correct answer also includes incorrect or irrelevant information. (I.e., don't put in everything you know in hopes of saying the correct buzzword.)

Question	Score
1-5 (25 points)	
6-10 (25 points)	
11-13 (40 points)	
total (max: 70 points):	

Stanford University Honor Code

In accordance with both the letter and the spirit of the Honor Code, I did not cheat on this exam nor will I assist someone else cheating.

Name and Stanford ID:

Signature:

Short answer questions: in a sentence or two, *say why your answer holds.* (5 points each).

1. You change Eraser: when it detects an access to memory location m will cause the lockset l associated with m to become empty, it instead acquires the locks in l and releases them after the access. How effective is this at removing errors?

2. Your ex-140 partner rewrites Eraser so it tracks a lockset for each *bit* of memory. Roughly, what will happen to the memory overhead? For the machine used in the Eraser paper, can doing things this way actually cause Eraser to miss errors?

3. Your MESA code has two monitors, M1:

   ```
   entry foo1() { bar1(); }
   entry foo2() { bar2(); }
   ```

   And M2:

   ```
   condition c;
   entry bar1() { wait(c); }
   entry bar2() { signal(c); } }
   ```

   What can happen? Why does MESA work this way compared to the alternative?

4. What two fixes should you make to the allocation code given in the MESA paper so that it will work with their system?

5. Assume that on your machine a read always returns the value of the last write. If you compile and run the code below using a compiler similar to the one used in the Boehm paper:

```
          % initial: a = 0, b = 1
        thread 1  |    thread 2
 ---------------|---------------------
   r1 = a;      |   r3 = b;
   r2 = a;      |   a = r3;
  if(r1 == r2)  |
        b = 2;  |
 ---------------|------------------
```

Give the smallest and largest number of registers `r1`, `r2`, and `r3` that could hold the value "2" after this code runs (and explain how).

6. Assume you check this code using Eraser:

```
 Thread1          |           Thread 2
my_lock(l);       |          my_lock(l);
x++;              |          x++;
my_unlock(l);     |          my_unlock(l);
y = x + 1;        |
```

Assume further that the code actually executes and Eraser flags no error — what plausible thing could be happening? (Hint: think like the Boehm paper and make sure your answer causes Eraser to miss errors rather than flag them.)

7. An excellent question came up after class: for Superpages, Table 3: `galgel`, `C4` and `SP` have 0.00 as their entry in column "`4MB`." What does this mean? What is the likely explanation? (Note, this might be a little tricky: make sure you know what the number means.)

8. ESX: Figure 6 (the toucher thread experiment): let's say you eliminate both of the fast averages: where in the figure would this be worst for the toucher thread? Let's say you eliminate the slow averages, where would this be worse for the toucher thread?

9. You change the failure oblivious code to return a random value on out-of-bound reads:

   ```
   T failure_oblivious_read_read(T *p) {
      if(inbounds(p))
          return *p;
      else
          return random();
   ```

   Explain what will (likely) happen for the midnight commander bug as compared to the approach they use.

10. Give a four line or less C code snippet that contains an error that will cause problems without failure oblivious, but will work “acceptably” with it. (Explain why!)

Problem 11: Events (15 points) In the cooperative paper:

1. (2 points) How could their "compute function" `GetCAInfo` cause performance problems with an event or cooperative system?

2. (3 points) Why don't they believe it's easy to forget that `GetCAInfoBlocking` has been transformed to continuation passing?

3. (5 points) What determines how big the memory pointed to by `cont` is? When a thread context switches, does the memory it needs to save state grow at the same rate? Why or why not?

4. (5 points) Which approaches can run `GetCAInfoBlocking` "as is" with no changes and still work? (Note: make sure you state any assumptions.)

Problem 12: Superpages (10 points) A program executes the statement `*p = 1` and then:

1. (3 points) The superpage containing the memory `p` points to is demoted. Why? What are the most number of distinct subpages it will produce?

2. (3 points) The page containing the memory `p` points to is promoted to a superpage. Why?

3. (4 points) The object containing the memory `p` points is promoted to a `512K` superpage. Give two cases where the reservation could have been 4MB.

Problem 13: ESX (15 points) Consider Figure 8 in the ESX paper.

1. (5 points) In 8(a) what is strange about the `alloc` line going far above 1GB? How can this happen?

2. (5 points) Consider minute 40 in 8(c) (the citrix server). What is going on with the two peaks and one valley? Which line is driving which?

3. (5 points) At around minute 70 in 8(c) what is going on? Which line is driving which in this case and, in particular, what is the likely reason for `active` going down? (Hint: it may help to look at 8(d).)