Stanford University
Computer Science Department
CS 240 Midterm Spring 2011

May 3, 2012

!!!!!! SKIP 15 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
1-6 (30 points)	
7-12 (30 points)	
13-15 (25 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. A thread runs the following procedure:

```
void foo(void) {
        int flag = 0;
        do_work(&flag);
        ...
}
```

   Later on in the program after all threads have exited Eraser flags an error in the code:

```
void fact(int n) {
        return (n<=1) ?  1 : n * fact(n-1);
}
```

   What is going on? (Hint: don't think too much about the code inside of `fact`: it's what you don't see that is important.)

2. Eraser: what is the contradiction between Figure 4 and the text? Which is correct?

3. You run:

   ```
   eraser ./a.out
   ```

   If eraser emits no error messages, does this mean `a.out` has no errors? If it emits an error, does `a.out` have errors? If you then rerun `a.out`, will eraser emit the same errors? Concisely state why or why not, especially for the last question.

4. We missed an important point in class: what is weird about how Mesa allocates procedure call records? How does this help bound the storage used for thread stacks?

5. You like `signal()` and `wait()` but don't want to write code in a dead language, so build an implementation in `pthreads` with the following type signatures:

```
void signal(cond_t *condition);
void wait(cond_t *condition);
```

   Your cs140 partner says you're going to have problems if the only parameter they take is a condition variable. What else would you have to pass in? If you already standardized your interface what implementation hack could you use from the Mesa paper to make things work (possibly)?

6. Boehm: You glance through the `NFW-threads` standard and notice the sentence: "A standard conforming program must use exactly one lock." Which problems (if any) in Section 4 of the Boehm paper will this eliminate?

7. Boehm: as suggested in class, you define the semantics of a `volatile` variable `v` as giving two guarantees: (1) no additional loads or stores can be done to `v` other than what appear in the program text and (2) an access to `v` cannot be reordered with any other volatile access or lock call. Which problems (if any) in Section 4 would this fix?

8. Give two places where scheduler activations block without notifying the user.

9. An implicit but overriding principle of the superpage paper is *primum non nocere* ("first, do no harm") in that they try to never be worse than the base system. Give two examples of choices they made that satisfy this principle and one example that does not.

10. Rectangle A states that the superpage guys should have measured the increase in memory footprint from using superpages. Rectangle B states any difference should be negligible. Who is more correct and why?

11. In what way does ESX's transparent page revocation make guest OSes have a drawback of user-level threads? How could you modify the ESX/guest interface to mitigate this problem?

12. ESX, Figure 8: around the 68 minute mark: explain the causal connection between alloc, balloon, and active in (c) and (d). (I.e., which one is driving the others, and the order in which the others influence each in turn.)

13. Will a guest OS be more or less susceptible to livelock when running on VMware? Let's say you are running screend on the "unmodified" OS from the livelock paper and that VMware knows you are forwarding packets. What could it do to detect and prevent livelock?

Your system has two kernel threads A and B with a message queue Q between them. There is also a user level process C. Assume the system is under heavy load and does not use the techniques from the livelock paper: give two bad things you would expect to see. In *at most 40 words* give the give the **complete** livelock solution for this system (ignore quotas and grammar).

Problem 15: Native Client (15 points) Consider the code in figure 3:

1. (3 points) Give three instructions `inst_is_disallowed` would check for.

2. (3 points) From the code: do direct jumps have to be aligned to 32-bytes? Do they end the 32-byte blocks?

3. (3 points) What is the attack if you delete the check `Block(StartAddr[icount-2] != Block(IP)`?

4. (3 points) What happens if you delete the four characters: `else`

5. (3 points) If you compute `StartAddr - JumpTargets` what do you get?