Stanford University
Computer Science Department
CS 240 Midterm Spring 2011

May 16, 2011

!!!!!! 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
1-8 (40 points)	
9 (15 points)	
10 (15 points)	
11 (20 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. Your unmedicated ex-cs140 partner writes code where locks are acquired by one thread and then deliberately released by a different thread. You know he can't code worth a damn, so run eraser on his system. Why might this programming style give Eraser problems? Can you change Eraser to partially work around this problem?
   Eraser only removes locks from the current thread's lockset (at least nothing in the prose suggests otherwise). Thus, if thread A releases lock l which is in thread B's lockset, eraser will still believe B holds the lock when it does not and consider any accesses it does falsely protected. When a lock is released you could traverse all active locksets and remove it, but this still makes detecting races scheduler dependent.

2. You want to modify Eraser so that it records lockset and variable accesses across application runs. Since Eraser is scheduler invariant, if your scheme worked could it find any additional errors? What challenge will you have?
   Eraser only finds bugs on the paths that execute and may miss errors on these paths if the locations accessed are data dependent (a typical situation). Thus, if subsequent runs hit new paths or new memory locations then they could find additional errors. A big challange is that eraser works in terms of addresses for memory and for locks (it seems). These will change from run to run.

3. Explain how to modify Eraser so that it can detect priority inversion in the `pthreads` programs it checks. How could it correct it?
   From Mesa: priority inversion is a high priority A thread blocked on low priority thread C (e.g., because it holds a lock) while some medium priority thread uses the CPU blocking B. Eraser knows which threads have which locks. Thus, when a thread goes to sleep on a lock, it can find who the owner is and donate the sleeping threads priority to it (if higher).

4. You read the latest version of the `pthreads` standard and notice the sentence: "Structure fields within the same word must be protected with the same lock." What problem in the Boehm paper does this attempt to fix? Does it fix all cases?
   This is trying to fix the case where a compiler modifies sub-word structure fields using word-sized loads and stores. This will not prevent the case where globals are stored in the same word and get modified in this way (e.g. two chars that are not in the same struct, but are placed in the same word).

5. Which broken code in the Boehm paper would likely work if you ran it on a machine that could only load and store bytes?
   The case where the compiler stores all structure fields with a word sized read — without word sized reads and writes presumably it won't do such tricks.

6. Events are a bad idea, Figure 3: "the performance of Knot-C versus Haboob shows that events are a bad idea." What is the flaw in this argument? What in the figure makes this case more strongly?
   Knot-C favors existing connections, whiile Haboob favors accept. Thus, under heavy load, its performance can drop to the floor — it spends all its cycles opening new connections and none servicing new ones. Knot-A is a better set of data points for this argument since its connect policy is (supposedly) close to haboob's.

7. Your native client verifier sees the following code:

```
...
0x12221004  mov a0, r2
0x12221008  call foo
0x1222100a  mov r1, r1
0x1222100f  add r1, r1, r2
...
```

   Will this code work correctly on native client? Why or why not? Hint: think about what happens with the call.
   The call is not the last instruction of a 32-byte block. Thus, when the nacl jump rips off the lower bits on the indirect return control will skip the rest of this block (`mov add`*) and got the beginning of the next one. If these instructions were needed then the result will be incorrect. A common mistake was to say that the call should have been a nacl jump, however this is a direct jump.*

8. In the x86 virtualization paper (Figure 1): when they adaptively retranslate a code block to avoid traps by inserting a new "simulated" block, they add a forwarding jump from the original block to the new block. What if they instead just removed the original block from the translation cache as well as its hash table entry so any continuations will automatically be translated to the new block. Does this work? Why or why not?
   Other translations can link to this block: remove it and they will still jump to this memory no matter what it is being used for. A lot of people mentioned the performance aspect of not being able to switch back to the original code when the block becomes "inocent" again. However this is only minor, you can always retranslate the block again and it doesn't affect the correctness.

Problem 9: Superpages (15 points)

Three points each:

1. (3 points) Give two programs in Table 2 that illustrate the problem they ran into with the Alpha's TLB miss cycle counter.
   Any two that have the same number of "TLB misses" but different speedups.

   Common mistake was to say that two columns with similar TLB misses but different speedups illustrated the problem — this isn't correct since translating miss reduction to speedup depends on how much of a percentage of total runtime was taken by misses. Perfectly accurate counters could have the same effect.

2. (3 points) Table 2: What is weird about FFTW? What type of access pattern must it be doing?
   It only has speedups for 4MB pages. It must be having huge numbers of misses in the TLB, which requires touching pages with a large stride so that it often accesses more than 128 512K superpages (128 = the D-TLB size) before it repeats.

   Common mistake was to say that it allocated large contiguous chunks of memory and then accessed them sequentially. If access was sequential, 512K superpages would have given a big speedup.

3. (3 points) If you could pick a single superpage size, what would it be and why? How much difference would you expect in their results in Table 1?
   512K. Should work pretty good. Superpage size doesn't matter much for most applications (only 1/3 appear in table 2 b/c they said the others showed a negligible difference). And for table 2, most of the win comes from that other than for a few apps.

4. (3 points) You compute the speedups in Table 1 as an average of the four runs they did rather than discarding the first one: what do you expect to happen?
   Less speedup. I/O (paging in data and text) will account for more of time spent, meaning that reduction in TLB overhead will be a smaller percentage of overall runtime.

   A common mistake was to say that the TLB would have more misses — but TLB entries are not re-used across processes.

5. (3 points) Assume you changed the Navarro promotion scheme to allocate an entire reservation's superpage as soon as *any* page in the reservation was allocated. If you reran the experiments in Table 1, what result do you expect?
   Shouldn't make much difference for Table 1: this is a best case, where memory was plentiful and no request fails and a warm-up run paged everything in. There should be plenty of free memory to absorb the extra bloat.

   We did give points if you said it would lead to a slowdown since it'd have to page stuff in or if you said the PTE replication would add overhead.

   A common mistake was to think that it would allocate 4MB superpages and so would blow up memory with massive overheads. Changing the promotion strategy has no impact on the rules they used for superpage reservation size — the sizes would remain the same as before.

Problem 10: Detecting errors (15 points) Assume you have a tool that can analyze MESA code.

1. The MESA paper sketches several deadlock scenarios. Explain how to use the tool to detect potential dead locks involving signal and wait.
   You want to see where monitor A calls two routines in monitor B, one of which does a singal and one of which does a wait — if two threads follow these two callchains, they can deadlock.

2. How could you detect which monitors will not work well with exceptions? Please state what you are looking for.
   Assume that monitor A calls monitor B. Recall that if a routine in B throws an exception that B does not catch, B's lock will not be released. No one can call into it anymore. Your tool could simply warn if a monitor (or a monitor it calls) can throw an uncaught exception.

3. You update the tool to understand C code. Give two errors that show up with cooperative threads or events that you could use it to detect (and how).
   Calling blocking I/O in both.

Problem 11: ESX (20 points)

1. (5 points) Figure 2: Why is the grey bar smaller than the black? What's a case where the black bar could be plausibly smaller than the grey?
   The guest OS has been configured with exactly the amount of memory it has — less overhead, thus more memory to go to buffer caches and etc, which is useful for the benchmark's performance. If the guest OS made bad paging decisions, giving it a lot of memory so that it never paged (and ESX did) could be better. We also accepted variants on "cache effects."

2. (5 points) Figure 6: why use a microbenchmark rather than a real application (or set of them)? What in the figure gives a measure of whether ESX is making good decisions for the application and the system?
   The allocation versus how much is being used. Good decisions for

   application = don't allocate less than needed. Good decisions for the system = alloc tight to what is needed.

3. (10 points) The VMware people want to put Google out of business and hire you to hack ESX (which runs on the bare hardware) so that it can be used to give the same security guarantees as native client. Sketch how you would do so. Assuming you get things right — what arguments could the google people (legitimately) make that your approach is worse than theirs?

   All native client is trying to do is to prevent the downloaded code from calling a small set of system calls (e.g., those to read and write the network or file system). It's trivial to add a small subsystem to ESX that will simply prevent a given process from issuing these calls — ESX can catch all system call traps and decide whether to forward them to the underlying OS or not. It will have to have some notion of what process is running. The google people would probably make the argument that: (1) the TCB will be huge, (2) there may be a non-trivial performance hit and (3) it requires clients run a VM rather than a unprivileged user level process.