# Stanford University
# Computer Science Department
# CS 240 Quiz 1
# Fall 2005

November 18, 2005

This is an open-book exam. You have 50 minutes to answer ten out of twelve questions. 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 - 4 | |
| 5 - 8 | |
| 9 - 12 | |
| total | |

**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:

Answer 10 of the following 12 questions and, in a sentence or two, *say why your answer holds.* (5 points each).

1. "MESA has monitored records." Give the complaint in the ERASER paper that this refutes.
   *The eraser guys whine that "Monitors...do not protect against data races in programs with dynamically allocated shared variables." This, of course, is complete nonsense for Mesa monitors.*

2. Give the intuitive reason the code in Boehm 4.3 behaves incorrectly. What will happen if you run this code through Eraser? What about the code in 4.1?
   *Main problem: loads and stores have been pulled outside of the critical section, completely destroying any guarantees (one of the symptoms in this case is that old values will get flushed back). Eraser will give an error since it will monitor at the level of assembly loads and stores rather than at the source code. The code in 4.1 will also cause Eraser to complain.*

3. Assume you can define a "`yield(...)`" routine that will save the state of the current thread. Rewrite `GetCAInfoBlocking` to use both `yield` and asynchronous I/O. This code should be very short. Define what `yield` must do after it saves the current process state in order to make your rewrite work.
   *Make sure you pass the event to yield. It has to couple the current thread to it so that it goes from blocked to unblocked. It must then get the next thread to run, perhaps by creating one and going back into the event loop. A common mistake was randomly yielding, but not making the disk read asyncronous.*

4. Assume you have the following code:

   ```
   unsigned char a;
   unsigned char b;
   ```

   Explain how to (1) correctly protect `a` and `b` according to pthreads, but get wrong answers and make Eraser emit an error, and (2) correctly protect `a` and `b` according to pthreads, give right answers, and not have Eraser emit any error. (You may have to make additional assumptions.)
   *Have a lock for* `a` *and different lock for* `b` *that you always acquire before every access and release after completing access (this satisfies pthreads' requirements); eraser protects 32-bit quanties so will see inconsistent locks being used for the word that contains a and b and give an error, on a machine that does not have byte level loads and stores a and b might get loaded as a unit and written back as a unit, violating atomicity. (2) if you use the same lock for both, both eraser and correctness will be satisfied (modulo any other variables that live next to a and b).*

5. Iron-chef VMS: Draw a graph that has a curve that starts low and goes high. Now state a useful, reasonable measurement you could have done on VMS that would look like this curve (label the X and Y axis appropriately) and why you believe this is plausible.
   *One easy possibility: The overhead of rescues as a function of figure 1. X-access is the same, y-axes is runtime, or the amount of time doing rescues.*

6. A program stores to address 0x410 on VMS; the TLB does not have an entry for this address. Explain all actions and checks the VMS hardware will do at this point. Be very concrete; include any errors that it could complain about.
   *This is a long one. First will access p0 page table. If outside bounds = error. If inside bounds will look up in PT. If valid bit not set = error; will give OS page fault and it can either fetch from disk/rescue list or give user a fault (if address just invalid). If protection not allow write = error, give to OS. Otherwise load value and put into TLB. Might miss at this point and then have to do a fault in system PT. A common mistake: assuming hardware looked on rescue lists, etc.*

7. "Because memory is plentiful there should be no difference." Give a reasonable question about the superpage paper that would have this as an answer.
   *Posisble: if you did promotion eagerly. Perhaps: if you ran the benchmarks concurrently.*

8. Your stack is 93K. What is your reservation size? What page sizes could the TLB entry for your stack have?
   *The object is dynamic, so reservation can go beyond the end. Therefore reservation size = 512K. Page size could be 8K (if memory fragmented) or 64K (if you could get the 64K superpage).*

9. You have two processes A and B. When they are 75% active you want A to have four times the memory of B. Give the taxation you would use. For this taxation, what is the largest and smallest this ratio can be?
   *This was a bit of a trick question. We took a variety of answers.*

10. The VMS guys like rescue lists and decide to hack them into ESX. Explain how this would work, which eviction decisions it would make the most sense to use these for, and give a reasonable way that such lists might plausibly improve performance.
    *Most likely place: put in where ESX randomly selects a page to evict. This could be a very bad decions, so makes sense to hold on. Assuming guest OS is smart = you probably don't want to do this as much for ballooned memory. A reasonable workload would be a large, active working set that ESX kept picking random pages from.*

11. Explain how Nooks' handling of parameters could cause a problem similar to one talked about in the Boehm paper.
    *It copies parameters back out of device. If this was data that was concurrently being accessed (even with locks) writing out an old value much later in execution will likely cause bad things to happen.*

12. Say concretely what will happen if we compiled the following code with Rinard's failure oblivious compiler and ran it:

```
char dest[4];
const char src[3] = {'x', 'y', 'z'};
strcpy(dest, src);
```

*strcpy copies until you hit a 0 byte. In this example there is garbage after the third character of src so could overrun dst. Rinard's compiler will discard all writes after the 4th. Not only that, in this case it will essentially make dest be null terminated in the 4th character since the first "random value" it always returns is 0.*