Stanford University
Computer Science Department
CS 240 Quiz 1 Solutions

May 5, 2008

This is an open-book exam. You have 50 minutes to answer **six of eight short questions** and **two of three long 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
1-8 (30 points)	**30**
9-11 (20 points)	**20**
total	**50**

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

1. 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?

   Deadlock:

   ```
   Thread 1      Thread 2

   M1.foo1();
   M1.foo2();
   ```

   M1.foo1() will sleep in M2.bar1(), but M1's monitor lock will not be released. If the only other call into M2 is through M1.foo2(), then we can never run signal. If M1's lock was released on a wait in another monitor then you would have to guarantee the monitor invariant before any external call. Such mistakes are harder to diagnose than deadlock, but the design decision was primarily about modularity.

2. Consider the manual/automatic stack management and serial/cooperative/preemptive labels in Figure 1 in the Cooperative Threading paper by Adya et al. How would you label:

   (a) The threaded programs checked by Eraser.:
   Automatic, pre-emptive.

   (b) A Mesa program.
   Automatic, pre-emptive.

   (c) The commands "`emacs foo.c; gcc foo.c`"
   Automatic, serial.

3. Assume that on your machine a read always returns the value of the last write. If you compile and run the code below using the compiler 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).

   Zero registers hold 2 if thread 2 runs to completion first. All three can hold it if the compiler optimizes. There are no lock/unlock calls, so it's perfectly legal for the compiler to redo thread 1's code to remove the if-statement (which is tautological for sequential code) and hoist the assignment:

```
 b = 2;
 r1 = a;
 r2 = a;
```

   Then the following interleaving gives all 2's:

```
 b = 2;
------->
r3 = b;
a = r3
<-------
 r1 = a;
 r2 = a;
```

   And there you have it.

4. BVT: re-draw Figure 1 for an idealized scheduling system that has no error in its proportional sharing. Assume you have 1000 processes, each with one share apiece, explain how they will be scheduled and when the worst error will occur.

   Straight line, all processes overlaid. Worst error: run 999 and then the moment before you run the first instruction in the 1000th. At this point we've run for 999 units of time, yet the 1000th process got 0 instead of (say) 1/1000.

5. You access a page not in your resident set on VMS — give the minimum and maximum number of pages it could be *required* to read from disk and why you decided on these numbers (Hint: max is at least two.)

   Min = 0: could be rescued or demand 0 filled. Max = two: the page itself could be paged out, as could the page table. Since the page table is flat this is just one page. It seems as if this is the max: the system page table cannot be paged, and none of the paging code can be paged. Their page clustering optimization could lead to more pages being pulled in, but this is not *required*; we gave points for noting that at least two may be needed.

6. The VMS guys want to adapt ESX's proportional share to improve VMS's resident set limit. While they adore ESX, they hate periodic sampling so want to use either page rescues or resident set evictions to estimate idleness — how would you do so and which is better? (Hint: think about big arrays.)

   Would have min (RSL) and max: processes guaranteed to get RSL, but can steal from the difference. Estimate idleness by shrinking their set and compute idleness from rescues. None = totally idle. All rescued = 0 idle. The problem with using evictions instead is that it seems less robust to blowing out the cache; if you iterate once over an array larger than memory, you will evict everything, but is useless to grow set since will not get reuse. Rescues indicate you need more memory and that you can exploit it (reuse makes caching work).

7. What are two reasons that a superpage system could perform worse than a non-superpage system despite the extra-ordinary measures the superpage implementors took to guard against this?

   Management overhead, paging decisions (which bias towards continuity at the slight expense of ejecting file pages earlier), and fragmentation costs.

8. You run the `matrix` benchmark in the superpage paper on the "Comparison of virtualization..." system with the important change that you use very small matrices. Give one argument *each* for why you may expect: (1) software virtualization to outperform the hardware approach and (2) there to be no difference.

   Key thing: small matrix = few TLB faults, PTE modifications, etc and that it's going to spend most time in user-space. Argument 2: User space code should run the same on both system. Argument 1: If you make the matrices small enough, the process isn't doing anything, so process creation and destruction will dominate. This should look more or less like the fork/wait microbenchmark on which software vm worked better.

Problem 9 You are looking at code in the Adya et al "Cooperative Task Management" system and see the routine:

```
 void equal(CAID ca1, CAID ca2) {
return GetCAInfo(ca1) == GetCAInfo(ca2);
 }
```

To show off you decide to rewrite the code to use `GetCAInfoHandler1` instead of `GetCAInfo`, in such a way as to maximize concurrent I/O requests. Sketch how the code looks. (You needn't give every semi-colon but there should be enough detail to distinguish your answer from someone who doesn't understand events and continuations.)

Ugh. Bad idea to ask this question. Basic idea: kick off I/O for the two events, rip equal in half.

Problem 10 Assume you have a runtime routine, `reachable(v)`, that Eraser can call that magically tells it there was any possible execution of the program that would allow a different thread to concurrently observe the memory associated with `v`.

(a) Explain how to adapt Eraser's state diagram (Figure 4) to use `reachable`: please be very concrete about any modifications you make with respect to error transitions or lockset refinements.

Before any lockset refinement, see if the variable is reachable. If it is not, don't refine (locks spurious).

(b) Give an example false positive from the case studies that your brave new algorithm would eliminate.

List ... locked list, got head pointer, nulled out list head, released lock and then accessed rest of list w/o lock.

(c) Assume `reachable(v)` instead just indicates if a thread *at this moment* can read `v` — does this change things in any significant way?

Yes. Means that it's not scheduling invariant: can miss races.

Problem 11 Suppose you have three virtual machines, A, B, and C, running on top of VMWare ESX. Each configured so the guest OS sees 100 MB of memory (meaning the VM's *max size* parameter is 100 MB). There are 180 MB of physical machine memory available to partition among these virtual machines after VMware's own data structures. All three machines run different guest OSes and experience minimal page sharing.

Machine A is more important than the other two machines, and is configured with $S = 4$ shares for memory allocation, while B and C have only one share each. On boot-up, all three machines touch all 100 MB of memory, but then A and B become completely idle, while machine C continues to use all of its memory. Assume the idle memory tax $\tau = 75\%$. How much memory will end up allocated to virtual machine A? Explain your reasoning.

80 MB.

The system will tend to equalize $\rho = S/P(f + k(1 - f))$ for each virtual machine. The idle page cost $k = 1/(1 - \tau) = 4$. For virtual machines A and B, the fraction of active pages $f = 0$, while for C $f = 1$.

Let P_A, P_B, and P_C be the number of pages allocated to A, B, and C respectively. Setting ρ of A and B to be equal, we have $4/(P_A \cdot 4) = 1/(P_B \cdot 4)$, so $P_A = 4P_B$. Similarly, $P_A = P_C$.

Since $P_A + P_B + P_C = 180\ MB$, we have that $P_A + (1/4)P_A + P_A = 180\ MB$, or $(9/4)P_A = 180MB$, or $P_A = 80MB$.