Stanford University
Computer Science Department
CS 240 Quiz 1
Spring 2004

March 9, 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. In Section 4.3 in the Eraser paper, they state that the routine `GmapCh_Write2` passes the address of a stack variable to its children threads. If this variable is on the stack, why does Eraser flag an error? Why can't they annotate this false positive away?

   Eraser only ignores locations explicitly indexed off the stack pointer; otherwise it tracks them — passing the address of a var means that it won't be indexed off the stack pointer. The FP happens because of memory reuse: stack memory is re-used on different function calls, but eraser doesn't notice this and so doesn't clear its bookkeeping data structures. they would need to tell eraser each stack record was being reused, but they don't seem willing to do so.

2. Assume only one thread has a pointer to a data location `p`. How would you use this ability to improve Eraser? (Make sure

   Initialization: start giving errors as soon as multiple threads could reach (this eliminates a class of false negatives). You should only give errors when multiple threads can reach a location; this elimanates numerous of their false positives. Some students ignored the fact that you can still end up with pointer aliasing problems

3. Thread 1 consists of a single call to `foo`, which is an entry routine in the following monitor M:

```
---------------------- Thread 1 -------------------------
void thread_1(void) {
      foo();
      bar();
}
---------------------------------------------------------

---------------------- Monitor M ------------------------
condition c;

entry foo() {
```

```
        wait(c);
        return;
}

entry bar() {
        signal(c);
        return;
}
```

Assume you are running on a MESA system and there are other threads on the system. Point out four locations in the code (in either the caller or callee) where thread 1 could get switched out. NOTE: the switches cannot occur for identical reasons.

(a) *Can get switched out at any point if another higher priority thread is running*

(b) *Will get switched if it calls M when someone else already holds M's lock.*

(c) *Will get switched if it returns and releases M's lock, and a higher priority thread is blocked waiting for the lock.*

(d) *Will get switched if it calls signal and wakes up higher priority (assumption: woken up thread gets put on run queue; this is not true if it gets put on monitor lock queue).*

4. The capriccio paper (weirdly!) limits the buffer cache in its experiments to 200MB. Assume you rerun the experiment without this limit and notice that LinuxThreads improves somewhat and capriccio performs much worse. What is a plausible explanation?
If you have a larger buffer cache, you would plausibly see more buffer cache hits. However, as more memory is used to cache files, there will be relatively less available for virtual memory caching. This measns you can plausibly have more page faults. Capriccio only handles explicitly blocking system calls, it does nothing for loads or stores that page fault. In this case, the entire capriccio process will be blocked and no capriccio threads will make progress.

5. The experimental hypothesis of Figure 3 in the VMS paper is not well evaluated. Give three *important*, concrete limitations and how to fix them.

   (a) *They only do one application, simulated. Better to run SPEC for real.*

   (b) *Do not measure paging overhead, need to figure out total cost of the increased rescuing that happens more and more memory devoted to free list.*

   (c) *Want to know the real impact on runtime.*

   There were lots of accepted solutions here, the above is just an example.

6. You use the shares-per-page ratio formula from the ESX paper to adjust the fixed partitions in VMS. However, you change the estimation of f to be based on the number of pages rescued over a 30 second period. Under what two conditions will this be identical to the original VMS scheme? In general, do you expect this estimation of f to work better or worse than the ESX estimation approach?

   This wasn't a very clear question... Sorry about that. Basically if you gave the correct formulas, taxation amount, etc you got credit.

7. Superpages are demoted both under memory pressure and when the first write occurs. How does this demotion differ, why does it differ, and when does re-promotion occur?

   Memory pressure: demote to base pages. Write: demote to largest subset of papers. Differs because memory pressure wants to get a fine grained view of what is in use. Repromotion occurs in first case when all pages referenced. In second, when all are marked dirty.

8. Your nooks extension makes an XPC call to a kernel function:

   ```
   foo(p);
   ```

and the nooks system restarts your extension. Give three different errors that could have occurred in the call to cause this restart to happen.

(a) p could be deallocated.

(b) foo modifies p but mapped r/o.

(c) p could be a pointer that gets freed when the extension returns, but foo could demand long term pointers.

9. You use the Safe-C compiler in the Rinard et al papers to implement memory protection in Nooks. What changes would you anticipate in Figures 6 and Figures 7 in the Nooks paper (and why!).

   Safe-C gives the ability to detect fine-grained illegal reads or writes, so this would allow us to detect more faults, reducing the number of non-fatel extension failures and raising the number of system crashes for the non-nooks (each detection = reboot system).

10. The failure oblivious guys try to put the nooks guys out of business by applying their approach to a network driver and a disk driver. In which would you expect to see better/safer results and why?

    Network should be much better: stateless device, and the code that uses it already does retransmission and checksumming to recover from lost data and detect corrupt data. Disk should be worse case: disk is dealing with presistent state, and writing out bad data to disk is generally not a hot idea.

11. Why does WFQ run the job with the minimum VFT rather than the job with the minimum VT?

    A process with few shares can then dramatically overshoot its fair share, giving large error. Extreme: A and B have the same VT, A has 1000 shares, B has 1 share. Doing B first will give huge error. Doing by VFT gives the smallest possible increase in error.

12. What in the WFQ scheduling pattern in Figure 5 is it impossible for VTRR to mirror?

WFQ can run processes in any order. VTRR can only run processes in the order they are initially sorted in.