CS240
Advanced Topics in Operating Systems
Quiz 2 – Spring 2008

SOLUTIONS

!! You should skip 10 points worth of questions !!

Your name: _____

Stanford ID: _____

In accordance with both the letter and the spirit of the Stanford Honor Code, I did not cheat on this exam. Furthermore, I did not and will not assist anyone else in cheating on this exam.

Signature: _____

The quiz has 12 questions. You have 50 minutes to complete 10 of them. Some questions may be much harder than others.

I Short answer

1. [**5 points**]**:** Your C runtime provides a function:

```
// Return object (if any) that addr is contained within: base holds
// start, size the object size in bytes.  Returns 0 if addr is not
// within any object.
int getobject(void *addr, void **base, unsigned *size);
```

You replace the object tracking code in the failure oblivious system and rewrite it to use `getobject` on every load and store to determine what object, if any, a pointer is contained within. Will the failure oblivious system likely work better, worse, or no different? (Make sure you state why!)

Will work worse. The failure-oblivious compiler tracks when a pointer goes out of bounds from the object it was intended to point to even if the pointer then points to a different (wrong) object. The `getobject` *procedure will not do this, so will allow garbage writes to occur that the failure oblivious compiler would block. E.g.,:*

```
p = malloc(10);
q = malloc(10);
p += 10000;  // goes out of bounds, but assume now points to q
*p = x;      // since this points to valid object, write succeeds.
```

Here the pointer `p` *goes out of bounds far enough to point to the object associated w/* `q`*. The write succeeds, when it should not.*

2. [**5 points**]**:** Nooks: You see the following code in a driver:

```
lock();
p->y++;
unlock();
```

If `p` came from the kernel, what is broken about this code? How would you have to fix it?

Problem: `p` *gets copied into the driver when we enter the driver and copied out when we leave (if modified). If these actions occur without locking then we have a race. Similarly, the locks above make no sense, since it's a private copy. Finally, we will be in a last-writer-wins situation where the flush back overwrites any preceeding writes. None of these actions will usually cause happy things to happen.*

3. [5 points]: Nooks: Violating linux religion, you write a bunch of device drivers that use a lot of (well-crafted and well-placed) `assert` statements and rerun the experiments in Figure 6 and 7: what changes might you plausibly see and why?

The hint was in the paper, just below the figures: "While Nooks is designed to protect the OS from misbehaving extensions, it is not designed to detect erroneous extension behavior." With the assertions, we now turn a lot of uncaught errors into assertion failures (presumably crashes); as a result, we detect more things and can restart in more cases, hopefully getting to a clean state.

4. [5 points]: LFS: What ordering rules do you have between writing out segments and checkpoints? Assume you crash and read in a valid checkpoint. What else do you have to do before you can start using the file system?

Better write out segment before writing out any checkpoint values that depend on segment (e.g., the imap). After a crash, you read in the checkpoint then have to roll forward any subsequent actions. (You could, also, just start immediately, and not worry about losing 30s of work.)

5. [5 points]: LFS: The segment usage table records the number of live bytes in a segment. You write a file. Explain how to update the segment table — which entries you would have to update, and how you would determine them.

When you write a file, you turn the segment space used to hold the old, overwritten contents into garbage. You have to find these segments and update their usage. This is pretty simple: whenever you move something and (therefore) overwrite a pointer, just decrement the corresponding space from the segment the old pointer pointed inside.

6. [5 points]: Sketch the eXplode checker (the mutate and check method) for the CVS check that the paper describes. Make sure you provide the intuition for what you are doing!

Basic sketch: the mutator should add/update some files, then do a "cvs commit" and then after issuing this command, call `check_crashes_now`*. The checker should make sure all the committed values are there.*

7. [5 points]: Assume it's the 1980s, so computers are too slow to efficiently do SHA1 or compression tricks. Sketch how you would retrofit two ideas from LBFS onto NFS (not compression and not hashing!) in a way that would clearly improve performance or functionality.

Pipelining (asynchronous writes), leases instead of the timeout caches, and atomic file updates would all be nice.

8. [5 points]: You run NFS on top of xsynfs instead of ext3. What kind of performance improvement do you expect to see compared to the experiments in the paper? (Justify your answer.)

Performance will probably not change that much: xsyncfs will have to block before sending a network message (externally visible event). Assuming the networking stack is not super slow this should be a negligible effect. If the NFS server is on a local machine, however, we may get major wins. Some of you assumed you could also make changes to NFS, and we gave reasonable credit for reasonable answers.

II "I am only sorry when I am caught."

xsynfs shows the power derived from doing anything you want as long as it can't be observed. Assume we change failure oblivious to exploit a similar dynamic. We mark registers and memory locations that contain fabricated values as "tainted".

9. [5 points]: How might you change the OS to use these tainted annotations so that it does not make "made up" values visible to the user? What do you have to do on memory deallocation, both from `malloc` and function call return? (Hint: think about eraser and memory allocation.)

The OS could just kill a process that tries to make tainted data visible. On memory reuse you need to clear tainted bits. The approach above is a bit sketchy since we either ignore control-dependencies (in which case can make taint-influenced stuff visible) or mark them (in which case almost everything becomes tainted.) Example:

```
if(x)
y = 1;
```

If `x` *is tainted, then if we care about control-dependencies we should taint* `y` *since* `x` *determines its value.*

10. [5 points]: Give two scenarios: one where you would work better than the original failure oblivious system, the other where you would work worse.

Worse: false positives. We might kill processes even when the visible output wasn't going to be visible (off the screen) or didn't matter (unresolved symlink replaced with some null string). Better: we will be sound (modulo the control dep. issue above) in that we immediately kill a process rather than letting bad data escape; for some application domains, this is important.

III Receive livelock

Mogul and Ramakrishnan re-architected the Digital Unix kernel so as to eliminate the livelock problem. The following pseudo code corresponds to the polling thread they describe:

```
poll ()
{
  int io;

  for (;;) {
   foreach (device) {
     if (clock_interrupt) {
        clock++;
        io = 1;
        for (i = 0; i < nthreads; i++) {
          if (thread[i].queue is full) {  // should thread i run?
            io = 0;
            break;  // let the scheduler schedule a user-level process
          }
        }
     } else {  // network device
       for (maximum of n pkts on device) {
         switch (type) {
         case RECEIVE:
           do device-specific stuff (e.g., copy packet from card);
           process_receiveintr(pkt);
           break;
         case SENDCOMPLETE:
           do device-specific stuff;
           process_sendcomplete(pkt);
           break;
         }
       }
     }
   }
   if (io) enable_interrupts(); // enable network interrupts
   wait (polling);
  }
}
```

11. [5 points]: What problem does the statement

```
for (maximum of n pkts on device)
```

in the function poll() address? Be concise and brief.

This is just the quota (timeslice) discussed in the paper: a simple-minded solution to ensure the rest of the system does not **starve**. *We accepted most conjugations of that verb.*

12. [5 points]: What problem do the following statements

```
if (thread[i].queue is full)  // should thread i run?
  io = 0;
  break;   // let the scheduler schedule a user-level process
}
```

in the function poll() address? If you omit the statement, what problems can occur? Be concise.

This is the feedback mechanism where we turn off interrupt processing when a queue is full. In this case we were about to drop a packet (and start down a slippery slope to livelock).