Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.828 Fall 2017

Quiz II

All problems are open-ended questions. In order to receive credit you must answer the question as precisely as possible. You have 110 minutes to finish this quiz.

Write your name on this cover sheet AND at the bottom of each page of this booklet.

Some questions may be harder than others. Read them all through first and attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

THIS EXAM IS OPEN BOOK AND OPEN LAPTOP, but CLOSED NETWORK.

Please do not write in the boxes below.

I (xx/12)	II (xx/6)	III (xx/12)	IV (xx/19)	V (xx/14)	VI (xx/21)	VII (xx/14)	VIII (xx/2)	(xx/100)

Name:

I xv6 Logging

Ben Bitdiddle is looking at commit() in xv6's log.c:

```
commit()
{
  if (log.lh.n > 0) {
    write_log();     // (AAA) Write modified blocks from cache to log
    write_head();    // (BBB) Write header to disk -- the real commit
    install_trans(); // Now install writes to home locations
    log.lh.n = 0;
    write_head();    // (CCC) Erase the transaction from the log
  }
}
```

write_head() writes the log head to the disk and waits for the write to complete. Ben observes that write_head() takes a long time, and he wonders why commit() calls write_head twice. He modifies his commit() to eliminate the second call to write_head() (marked CCC).

1. [6 points]: Describe a sequence of events in which Ben's modification will lead to incorrect file system operation.

Ben restores the second call to write_head().

Now he notices that write_log() and write_head() end up writing the disk in an order that leads to low performance. On the disk, the log header block is just before the log data blocks, so commit() ends up having to wait for a whole rotation of the disk between the write_log() (AAA) and the write_head() (BBB). Ben swaps lines AAA and BBB, hoping to avoid the extra rotation.

2. [6 points]: Describe a sequence of events in which Ben's modification will lead to incorrect file system operation.

II EXT3 Logging

Recall Linux's ext3 file system, described in Lecture 14 and in Tweedie's paper *Journaling the Linux ext2fs Filesystem*.

Ben observes that ext3 writes meta-data blocks (i-nodes, directory blocks, free-block bitmaps, indirect blocks) to the disk twice, once to the journal and once to their home locations. Ben is considering speeding up ext3 by writing modified meta-data blocks to their home locations as soon as system calls modify them, and later writing the blocks to the journal.

3. [6 points]: Describe a sequence of events in which Ben's modification will lead to incorrect file system operation.

III Lab 4

Ben is having trouble implementing the user page fault handler for copy-on-write fork in Lab 4: he is confused about how to get the right environment id to pass to the system calls.

4. [6 points]: Ben tries passing the global pointer thisenv to system calls. However, his child page fault handler incorrectly issues system calls to the parent's environment id. Assuming the rest of Ben's code is correct, what could be causing this bad behavior?

Ben's lab4 is still broken and `forktree` crashes with unexpected page faults. After much writing, re-writing, and debugging, he ends up with the following:

```
envid_t
start_fork(void)
{
        // set parent's page fault handler...
        return sys_exofork();
}

envid_t
fork(void)
{
        envid_t child = start_fork();
        if (child == 0) {
                // child stuff...
        } else {
                // parent: call duppage() to map the parent's pages in the
                // child, etc...
        }
        // finish fork and return
}
```

After tinkering with his code, Ben is astonished to find that his lab4 works perfectly if he moves the call to `sys_exofork` from the `start_fork` function to the `fork` function as follows:

```
void
start_fork(void)
{
        // set parent's page fault handler...
}

envid_t
fork(void)
{
        start_fork();
        envid_t child = sys_exofork();
        if (child == 0) {
                // child stuff...
        } else {
                // parent: call duppage() to map the parent's pages in the
                // child, etc...
        }
        // finish fork and return
}
```

5. [6 points]: Explain why Ben's lab4 crashes when `sys_exofork` is called from `start_fork`. Assume that all other code is correct.

Name:

IV Virtual Machines

For these questions, refer to Adams and Agesen's *A Comparison of Software and Hardware Techniques for x86 Virtualization*.

6. [6 points]: The x86 `CLI` and `STI` instructions disable and enable interrupts respectively. They each take a dozen or so cycles to execute — the reasons for these times are not documented but probably have something to do with serializing the delivery of interrupts. Surprisingly, with VMware's software VMM, a guest kernel can execute CLI and STI instructions faster than it could on a native machine. Explain how VMware binary translation handles a guest's CLI and STI instructions.

7. [6 points]: For what purpose does code generated by VMware's binary translator use the x86 segmentation registers?

8. [7 points]: Suppose you wanted to port the VMware software VMM (and associated binary translator) to a machine without segmentation but otherwise like the x86 (for example, to a 64-bit Intel CPU). What could your binary translator do in place of using segmentation registers? Explain your answer.

V Virtual Memory

For these questions, refer to Appel and Li's *Virtual Memory Primitives for User Programs* and Belay et al. *Dune: Safe User-level Access to Privileged CPU Features*.

9. [7 points]: Ben Bitdiddle is working on a new programming language that uses a concurrent version of Baker's algorithm to perform garbage collection (outlined in Section 3 of the Appel and Li paper and the slide entitled "Solution: Use virtual memory!" in the Virtual Memory 2 lecture). In an effort to improve performance, Ben devises a scheme where he maps unscanned pages in the to-space as read-only instead of disabling boths reads and writes. Unfortunately, the program crashes as a result of this change. Assuming Ben makes no other changes, explain which Baker's algorithm invariant is being violated.

10. [7 points]: Instead of making changes to the garbage collection algorithm, Alyssa P. Hacker suggests that Ben could use Dune to handle page fault exceptions directly from userspace. After porting the language runtime to Dune, Ben observes a large speedup in garbage collection performance. However, programs that compute a lot but don't involve much garbage collection run a bit slower. What is the most likely explanation for this slowdown?

VI Synchronization

For these questions, refer to Boyd-Wickizer et al. *Non-scalable locks are dangerous* and McKenny et al. *RCU Usage In The Linux Kernel: One Decade Later*.

Ben is working on a version of XV6 with the goal of supporting efficient, flexible networking. So far he has implemented the following code to allow for routing parameters to be reconfigured while transmitting packets. Normally, a network stack would support several different routes, but in Ben's prototype only a single route is supported so far.

```
spinlock_t route_lock;
// information about the next router hop.
struct route {
        uint32_t hostip;
        uint32_t gatewayip;
        uint32_t subnetmask;
};
struct route *route;

// Transmit a packet.
void transmit_pkt(struct pkt *p)
{
        struct route r;
        acquire(&route_lock);
        r = *route; // make a copy
        release(&route_lock);
        transmit_on_route(p, &r);
}

// Update the routing information with new parameters.
void change_route(struct route *nroute)
{
        struct route *oroute;
        acquire(&route_lock);
        oroute = route;
        route = nroute;
        release(&route_lock);
        free(oroute);
}
```

11. [7 points]: XV6 normally uses test-and-set (TAS) locks, which are known to be susceptible to congestion collapse. Ben tries replacing XV6's locking scheme with MCS locks. He then tests the system's performance using a single-threaded network application. To his surprise, the CPU time spent in `transmit_pkt()` increased slightly after changing the locking implementation. Why are MCS locks slower in this situation?

Recall the Linux RCU API you learned about in lecture. A summary is as follows:

- **rcu_read_lock()** Begin an RCU critical section.
- **rcu_read_unlock()** End an RCU critical section.
- **synchronize_rcu()** Wait for pending RCU critical sections to finish.
- **rcu_dereference()** Signal the intent to dereference a pointer inside an RCU critical section.
- **rcu_assign_pointer()** Assign a value to a pointer that is read in RCU critical sections.

Using RCU, Ben reimplements `transmit_pkt()` and `change_route()` as follows.

```
spinlock_t route_lock;
// information about the next router hop.
struct route {
        uint32_t hostip;
        uint32_t gatewayip;
        uint32_t subnetmask;
};
__rcu struct route *route;

// Transmit a packet.
void transmit_pkt(struct pkt *p)
{
        struct route r;
        rcu_read_lock();
        r = *rcu_dereference(route); // make a copy
        rcu_read_unlock();

        synchronize_rcu();
        transmit_on_route(p, &r);
}

// Update the routing information with new parameters.
void change_route(struct route *nroute)
{
        struct route *oroute;
        acquire(&route_lock);
        oroute = route;
        rcu_assign_pointer(route, nroute);
        release(&route_lock);
```

Name:

```
        free(oroute);
}
```

12. [7 points]: Ben’s code has a bug in the way it uses RCU. Clearly explain the bug.

Name:

13. [7 points]: How will Ben's RCU bug cause his code to malfunction?

VII IX

You would like to use IX on your heavily loaded memcached server. Your server has a 40-gigabit/second Ethernet NIC, which supports a single pair of DMA queues (one incoming queue, one outgoing queue). Your server has eight cores, and you'd like to use all of them for memcached. You are thinking of modifying IX so that application threads on all eight cores share the single pair of NIC DMA queues.

14. [7 points]: Explain why this arrangement would likely deliver lower performance than if you used a 40-gigabit/second NIC that supported multiple pairs of DMA queues. Be specific about what would go wrong, and why.

Ben Bitdiddle has a multi-core memcached server that uses IX. The server has a NIC that supports many pairs of DMA queues. He is thinking of modifying the way his NIC chooses the queue on which to place each incoming packet. Instead of having the NIC hash the packet's port numbers and IP addresses in order to choose the queue, he wants to modify his NIC to place each incoming packet on the queue that currently has the fewest packets on it. You should assume that this modification is possible, and that it does not slow down the NIC. Ben does not modify IX. His memcached uses TCP.

15. [7 points]: Explain what will go wrong and why.

VIII 6.828

16. [1 points]: What was your favorite topic in 6.828?

17. [1 points]: Which 6.828 topic should we discard next year?

End of Quiz II