Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.1810 Fall 2025

Quiz II

You have 120 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.

Gradescope E-Mail Address:

Name:

I Threads

Ben Bitdiddle discovers that the C compiler he uses to compile the xv6 kernel generates code that never uses the `s11` register. He decides to optimize `swtch` by removing the lines that save and restore `s11` from `swtch.S`. However, Ben compiles the user-space applications running on top of his xv6 kernel with a different compiler that does make use of `s11`.

1. [10 points]: Would this be a safe change to make, assuming that the kernel itself is always compiled with Ben's compiler that never uses `s11`?

Name:

II Coordination

Ben is extending the xv6 kernel to support a new pipe-like abstraction, the *bipe*. A bipe has just one reading and one writing process, which are known in advance when the bipe is created. Only a single byte is ever written to a bipe.

Ben would like a bipe's reading process to wait for the one byte if the writer hasn't written it yet, and he'd like the reader to wait without spinning. He realizes that xv6's sleep and wakeup could be used here, but he believes he can use simpler code that's specialized to his use case.

On the next page is part of Ben's kernel bipe implementation (what's missing is initialization, freeing when no longer in use, and glue code to allow user system calls to use the functions). In Ben's application there's no need for a bipe reader to worry about being killed while waiting.

Name:

```
struct bipe {
  struct spinlock lock;
  char data;        // just one character
  int ready;        // has data been written?
  struct proc *dst; // the one reader
};

int biperead(struct bipe *b) {
  struct proc *p = myproc();
  while(1){
    acquire(&b->lock);
    if(b->ready){
      char c = b->data;
      release(&b->lock);
      return c;
    }
    release(&b->lock);

    // AAA

    acquire(&p->lock);
    p->state = SLEEPING;
    sched();
    release(&p->lock);
  }
}

void bipewrite(struct bipe *b, char c) {
  acquire(&b->lock);
  b->data = c;
  b->ready = 1;
  release(&b->lock);

  // BBB

  acquire(&b->dst->lock);
  if(b->dst->state == SLEEPING)
    b->dst->state = RUNNABLE;
  release(&b->dst->lock);
}
```

2. [10 points]: Ben's bipe implementation has a serious correctness problem. What is it?
(Circle the one best choice; we subtract points for incorrect answers.)

A. xv6 forbids kernel code from holding locks when calling `sched()`, but `biperead()` holds `p->lock`.

B. if `biperead()` executes entirely while `bipewrite()` is at line `BBB`, the reader may never see the data.

C. `biperead()` should hold `b->lock` for the entire time, acquiring it at the start of the function, and only releasing it just before returning.

D. if `bipewrite()` executes entirely while `biperead()` is at line `AAA`, the reader may never see the data.

E. `bipewrite()` acquires `b->dst->lock`, but it should instead acquire its own lock (i.e. `myproc()->lock`).

III Networking lecture/reading

Consider *Eliminating Receive Livelock in an Interrupt-driven Kernel*, by Mogul *et al.*, and Lecture 15.

Ben Bitdiddle runs xv6 just like in the net lab (with 3 CPUs), but configures the hardware to deliver the network card interrupt only to CPU 0. Ben's xv6 network stack does not use transmit interrupts.

3. [10 points]: Could Ben's system experience receive livelock? Briefly explain your answer.

IV Shenango

Which of the following are true statements about Shenango as described in “Shenango: achieving high CPU efficiency for latency-sensitive datacenter workloads”?

(Circle True or False for each choice; we subtract points for incorrect answers.)

4. [12 points]:

A. True / False To get high performance, Shenango allows the IOKernel to read and write descriptors in the NIC queues from user space.

B. True / False Shenango may run an application on cores guaranteed to another application.

C. True / False A 99.9% tail latency of 37μsec means 999 out of 1000 requests take more than 37μsec.

D. True / False Even when memcached is under high load from clients Shenango manages to run a batch processing application occasionally.

Name:

V File System Layout in xv6

Alyssa is using xv6 with triply-indirect blocks, and her file system has 10 million 1024-byte blocks (`FSSIZE` in `param.h` is 10000000). In one directory, Alyssa creates one file whose size is nearly 10 million blocks (so it almost fills the disk, but not quite). In another directory, that starts out empty, Alyssa creates a new file called `x` with this shell command:

```
echo hello > x
```

No other file-system-related actions occur.

5. [10 points]: Approximately how many blocks will Alice's `echo` shell command cause the kernel's file-system code to **read** from the disk?

(Circle the one best choice; we subtract points for incorrect answers.)

A. 2

B. 10

C. 600

D. 1300

E. 2600

VI EXT3

Recall the Linux EXT3 journaling file system from *Journaling the Linux ext2fs Filesystem* and Lecture 19. The paper's "ext2fs" is the same as EXT3, and we are referring to the file system running with ordered data (as opposed to journal data) mode.

6. [10 points]: Which of the following are true statements about EXT3.
(Circle True or False for each choice; we subtract points for incorrect answers.)

A. True / False EXT3 can combine changes from multiple system calls in a single log/journal commit.

B. True / False When writing 1MB of data to a file, EXT3 will write roughly 2MB of data to disk (writing 1MB to the log/journal, and then installing that 1MB to the file's data blocks).

C. True / False After a crash, EXT3 might be missing the last few file changes.

D. True / False In EXT3, a system call can be modifying a directory `/d` while EXT3 is in the process of writing previous changes to directory `/d` to its on-disk journal.

E. True / False EXT3 can evict cached disk blocks that have been committed to the journal but not yet installed.

Suppose you run the following application on EXT3, where, before this program ran, the files `/a` and `/b` never existed, and `ccc` never appeared anywhere on disk:

```
int main() {
  int fd;

  fd = open("/a", O_CREAT|O_RDWR, 0666);
  if (fd < 0)
    exit(1);

  for (int i = 0; i < 3; i++)
    write(fd, "a", 1);
  close(fd);

  fd = open("/b", O_CREAT|O_RDWR, 0666);
  if (fd < 0)
    exit(1);

  for (int i = 0; i < 3; i++)
    write(fd, "b", 1);
  close(fd);

  fd = open("/a", O_RDWR, 0666);
  if (fd < 0)
    exit(1);

  for (int i = 0; i < 3; i++)
    write(fd, "c", 1);
  close(fd);

  exit(0);
}
```

Assume the program exits with exit status 0, and the system crashes after that.

7. [12 points]: What could you see in the file system after the system boots up and recovers? **(Circle True or False for each choice; we subtract points for incorrect answers.)**

A. True / False You could see that `/a` and `/b` both exist and are both empty.

B. True / False You could see that neither `/a` nor `/b` exist, but `ccc` appears somewhere on disk.

C. True / False You could see that `/a` contains `aaa` and `/b` is empty.

D. True / False You could see that `/a` does not exist, and `/b` contains `bbb`.

VII Multicore scalability and RCU

Ben implements his read-write lock for the lock lab as follows:

```
struct rwspinlock {
 int32_t n;
};

void rwlock_r_acquire(struct rwspinlock *rwlk) {
 while (1) {
  int32_t v = __atomic_load_n(&rwlk->n, __ATOMIC_ACQUIRE);
  if (v < 0) { // does a writer have the lock?
   continue;
  }
  // __atomic_compare_exchange_n returns true if the value at &rwlk->n
  // matched v and the value was successfully updated to v+1.
  // It returns false otherwise.  You can ignore the last 3 arguments.
  if (__atomic_compare_exchange_n(&rwlk->n, &v, v+1,
                                  0, __ATOMIC_ACQUIRE, __ATOMIC_RELAXED)) {
     break;
  }
 }
}

void rwlock_r_release(struct rwspinlock *rwlk) {
  // atomically subtract 1 from rwlk->n
 __atomic_sub_fetch(&rwlk->n, 1, __ATOMIC_RELEASE);
}

void rwlock_w_acquire(struct rwspinlock *rwlk) {
 while (1) {
  // if rwlk->n is 0, set it to -1, and return true; otherwise
  // return false
  int32_t v = 0;
  if (__atomic_compare_exchange_n(&rwlk->n, &v, -1,
                                  0, __ATOMIC_ACQUIRE, __ATOMIC_RELAXED)) {
     break;
  }
 }
}

void rwlock_w_release(struct rwspinlock *rwlk) {
 __atomic_store_n(&rwlk->n, 0, __ATOMIC_RELEASE);
}
```

8. [8 points]: Which of the following statements are true about Ben's implementation?
(Circle True or False for each choice; we subtract points for incorrect answers.)

A. True / False The lock can never be held by a reader and a writer at the same time

B. True / False The lock can be held by many readers

C. True / False A writer may never be able to acquire the lock

D. True / False The lock can be acquired in read mode $2^{32} - 1$ times without releasing the lock

Consider a multithreaded program in which n threads acquire a read-write lock in read mode, compute for some time t, and then release the lock. None of the threads acquire the lock in write mode. The program is running on a 16-CPU x86 processor as used in the RCU paper "RCU Usage in the Linux kernel: one decade later".

9. [8 points]: Which of the following are true statements?
(Circle True or False for each choice; we subtract points for incorrect answers.)

A. True / False If n is large and t is 1s, the lock will be a performance bottleneck.

B. True / False If n is large and t is 0s, the lock will be a performance bottleneck.

Name:

VIII Containers and VMs

Consider the reading “Blending Containers and Virtual Machines: a study of Firecracker and gVisor (2020)”. Suppose an adversary discovers that calling `close(-1)` causes the Linux kernel to crash, and writes a program to invoke `close(-1)`. Assume there are no other bugs.

10. [9 points]: Which of the following are true statements?
(Circle True or False for each choice; we subtract points for incorrect answers.)

A. True / False If the program is running on gVisor, the program causes a crash in the host kernel shown in Figure 2, if the host kernel is the buggy Linux kernel.

B. True / False If the program is running on Firecracker, the program causes a crash in the Linux kernel shown at the bottom of Figure 3 (assuming it is a buggy Linux kernel).

C. True / False If the program is running in a LXC container on top of a Linux kernel, the program causes a crash in the Linux kernel (assuming `close` is on the allowed syscall list).

VEE '20, March 17, 2020, Lausanne, Switzerland

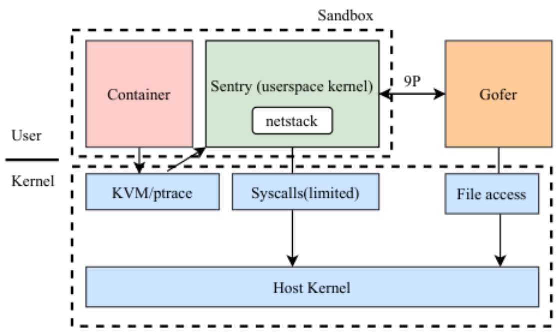

Figure 2. gVisor architecture

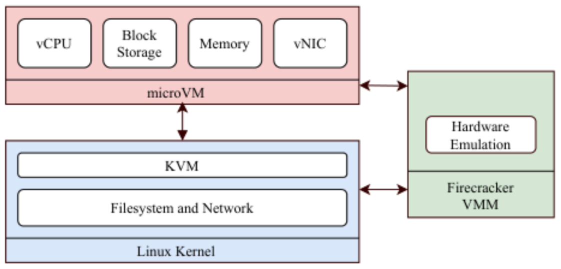

Figure 3. Firecracker architecture

Name:

IX Meltdown

The paper *Meltdown: Reading Kernel Memory from User Space*, by Lipp *et al.*, mentions that KAISER is an effective defense against Meltdown.

11. [10 points]: Why does KAISER prevent Meltdown from working?
(Circle the one best choice; we subtract points for incorrect answers.)

A. KAISER marks each page of physical RAM as inaccessible to user code.

B. KAISER clears the PTE_U bits of PTEs for kernel virtual addresses in user page tables.

C. KAISER causes kernel virtual addresses to all be mapped to physical page zero, so that Meltdown loads the wrong data.

D. KAISER causes user memory not be be mapped when the kernel is executing, so that kernel activity cannot affect user space.

E. KAISER enables kernel address space layout randomization (KASLR), which makes Meltdown impossible.

F. None of the above.

X 6.1810

12. [2 points]: What lab did you learn the least from (net, lock, fs, or mmap), and why?

13. [2 points]: Which of the papers (Livelock, Shenango, ext3, RCU, containers, BPF, Meltdown) do you think we should delete next year, and why?

End of Quiz II — Happy holidays!

Name: