HW12 Solutions

1.

This does not work. It violates the golden rule of atomicity (never modify the only copy). In more detail, there are certain operations that require writing to more than one place on the disk, for example, appending to a file (which might require writing to an indirect block, updating a bitmap, and updating the inode itself). If the machine crashes in between these different disk updates, then because the operation is not complete, the system never got to the point of writing a journal entry. Thus, the on-disk data structures are now inconsistent, and there is no indication in the logs. The recovery process as described does not detect or fix this issue.

2.

The code is not correct. Assume lock is unlocked, so locked == 0.

```
P1: enters exchange_value()
P1: int was = *ptr // P1's copy of was is now 0
P2: enters exchange_value()
P2: int was = *ptr // P2's copy of was is now 0
P1: *ptr = 1
P2: *ptr = 1
P1: return was // remember, was == 0
P2: return was // remember, was == 0
```

Then both threads break out of the spinning, so both continue, so there is no mutual exclusion in this case.

3.

```
typedef enum {FREE, IN_USE} key_status;

class Monitor {
    public:
        Monitor() { memset(&keys, FREE, sizeof(key_status)*5); }
        ~Monitor() {}
        void take_key(int desired_car);
        void return_key(int desired_car);
    private:
        Mutex mutex;
        key_status keys[5];
        /* ADD MATERIAL BELOW THIS LINE */

        Cond cond;
};

void driver(thread_id tid, Monitor* mon, int desired_car) {
    /* you should not modify this function */
    mon->take_key(desired_car);
    drive();
    mon->return_key(desired_car);
}

void Monitor::take_key(int desired_car) {
    /* FILL IN THIS FUNCTION. Note that the argument refers
       to the desired car. */

    int desired_key = desired_car / 2;

    mutex_acquire(&mutex);

    while (keys[desired_key] != FREE) {
        mutex_wait(&mutex, &cond);
    }

    keys[desired_key] = IN_USE;

    mutex_release(&mutex);
}

void Monitor::return_key(int desired_car) {
    /* FILL IN THIS FUNCTION. Note that the argument refers
       to the desired car. */

    int desired_key = desired_car / 2;

    mutex_acquire(&mutex);

    keys[desired_key] = FREE;
    cond_broadcast(&mutex, &cond);

    mutex_release(&mutex);
}
```