Assignment 2

ECSE 420: Parallel Computing

Due: March 14, 2024

Submission instructions:

Students are to work in groups of two on the assignment. Each group should submit a pdf file that contains the following information: students' names, students' IDs, instructions on how to run each file and the associated question solved. Students are also expected to submit a .zip file containing their source code. This code must compile and run without error. The code must be well formatted, commented, and follow the Google Java Style Guide. For more information, see the ECSE420AssignmentSubmissionInstructions.pdf in the Assignment section on myCourses.

Questions

1. **(24 marks)** This question considers *n*-thread mutual exclusion algorithms.
   1.1. Implement the Filter lock described in Chapter 2 of the course text.
   1.2. Does the Filter lock allow threads to overtake other threads an arbitrary number of times? Explain.
   1.3. Implement Lamport's Bakery lock described in Chapter 2.
   1.4. Does the Bakery lock allow some threads to overtake others an arbitrary number of times? Explain
   1.5. Propose a test that verifies if a lock works for *n*-thread mutual exclusion. Provide an implementation for the proposed test and verify if the locks implemented above do provide *n*-thread mutual exclusion.

2. **(8 marks)** Consider **LockOne** and **LockTwo** introduced in the textbook; do they still satisfy two-thread mutual exclusion if the shared atomic registers - "flag" in LockOne and "victim" in LockTwo - are replaced by *regular* registers?

3. **(12 marks)** Programmers at the Shaky Computer Corporation designed the ShakyLock protocol shown in Fig. 1 to achieve *n*-thread mutual exclusion. For each question, either sketch a proof, or display an execution where it fails.
   3.1. Does this protocol satisfy mutual exclusion? (Hint: Start the proof by assuming that two threads A and B are in the critical section at the same time.)
   3.2. Is this protocol *deadlock-free*? Explain.
   3.3. Is this protocol *starvation-free*? Explain.

```
class ShakyLock implements Lock {
  private int turn;
  private boolean busy = false;
  public void lock() {
    int me = ThreadID.get();
    turn = me;
    do {
          busy = true;
    } while ( turn == me || busy);
  }
  public void unlock() {
    Busy = false;
  }
}
```

Fig.1 ShakyLock lock protocol used in Question 3.

4. **(12 marks)** For each of the histories shown in Figs. 2 and 3, determine if are they *sequentially consistent* and if they are *linearizable*? Justify your answers.

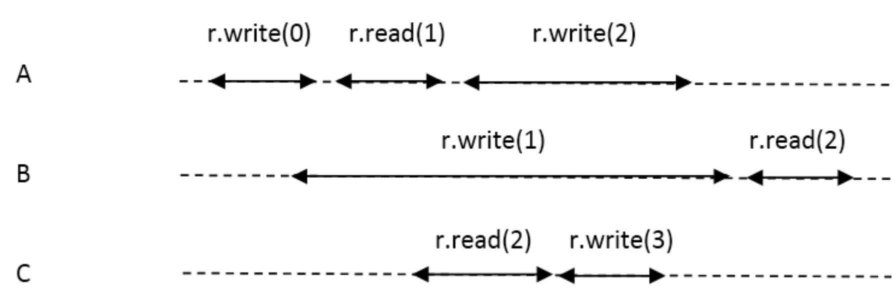

Fig. 2 History A

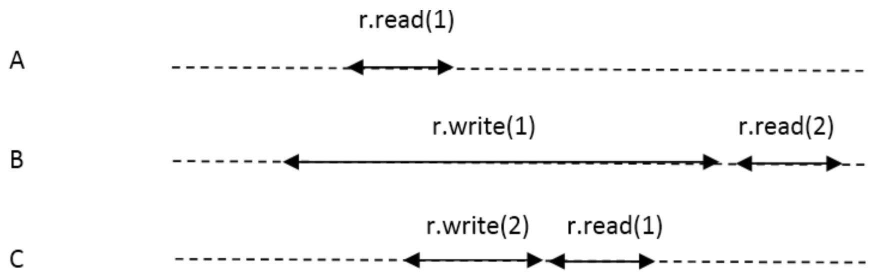

Fig. 3 History B

5. **(8 marks)** Consider the class shown in Fig. 4. Suppose two threads *A* and *B* are concurrently calling the methods *writer()* and *reader()*.
   5.1. According to what you have been told about the Java memory model, will the *reader* method ever divide by zero? If yes, describe the order in which *writer* and *reader* should be invoked (by threads A and B) and take effect for a division by zero to happen.
   5.2. Is division by zero possible if both x and v are volatile? What happens if neither *x* nor *v* are volatile? Justify your answer.

```java
class VolatileExample {
  int x = 0;
  volatile boolean v = false;
  public void writer() {
    x = 42;
    v = true;
  }
  public void reader() {
    if (v == true) {
      int y = 100/x;
    }
  }
}
```

Fig. 4 Volatile example

6. **(8 marks)** Consider the regular M-valued MRSW construction shown in Fig. 5; True or false:
   6.1. If we change the loop at line 11 to "for (**int** i = x + 1; i < RANGE; i++)", then the construction is still a *regular* M-valued MRSW register. Justify your answer.
   6.2. If we change the loop at line 11 to "for (**int** i = x + 1; i < RANGE; i++)", then the construction yields a *safe* M-valued MRSW register. Justify your answer.

```
1  public class RegMRSWRegister implements Register<Byte> {
2    private static int RANGE = Byte.MAX_VALUE - Byte.MIN_VALUE + 1;
3    boolean[] r_bit = new boolean[RANGE]; // regular boolean MRSW
4    public RegMRSWRegister(int capacity) {
5      for (int i = 1; i < r_bit.length; i++)
6        r_bit[i] = false;
7      r_bit[0] = true;
8    }
9    public void write(Byte x) {
10     r_bit[x] = true;
11     for (int i = x - 1; i >= 0; i--)
12       r_bit[i] = false;
13   }
14   public Byte read() {
15     for (int i = 0; i < RANGE; i++)
16       if (r_bit[i]) {
17         return i;
18       }
19     return -1; // impossible
20   }
21 }
```

Fig. 5 The regular M-valued MRSW class

7. **(4 marks)** Show that if binary consensus using atomic registers is impossible for two threads, then it is also impossible for n threads, where n > 2. (Hint: argue by reduction: if we had a protocol to solve binary consensus for n threads, then we can transform it into a two-thread protocol.)

8. **(4 marks)** Show that if binary consensus using atomic registers is impossible for n threads, then so is consensus over k values, where k > 2.

Total: 80 marks