Assignment 2 Solutions

Marco De Liso (18-120-261)

March 26, 2023

Contents

1 Task 1: Counter concurrent 1

2 Task 2: Read/Write implementation 2

3 Task 3: Linearizability and Sequential Consistency 3

1 Task 1: Counter concurrent

Consider the "Counter" program seen in the labs and provided on ILIAS with different synchronization mechanisms used to protect a shared integer. As a reminder, the program takes as arguments two integers, T and N, and forks T threads that modify a shared integer as follows. Even threads (i.e., threads 0, 2, 4...) increment the integer N times while odd threads (i.e., threads 1, 3, 5...) decrement it N times. The program prints the final value of the integer (which should be 0 if T is even, or N otherwise). Execute the program with N=10'000'000 and T=2,4,8,16 and all provided synchronization mechanisms. Report execution times.

- **Answer:**
- Input: N = 10'000'000, T = 2, 4, 8, 16
- Output:
  - Counter.java (no synchronized blocks)
    - Start with 2 threads:
    - Finished with total of 497513 in 163 ms
    - Start with 4 threads:
    - Finished with total of 665362 in 414 ms
    - Start with 8 threads:
    - Finished with total of -1107673 in 840 ms
    - Start with 16 threads:
    - Finished with total of -1621931 in 1662 ms
  - CounterAtomic.java
    - Start with 2 threads:
    - Finished with total of 0 in 188 ms
    - Start with 4 threads:
    - Finished with total of 0 in 420 ms
    - Start with 8 threads:
    - Finished with total of 0 in 886 ms
    - Start with 16 threads:
    - Finished with total of 0 in 1782 ms
  - CounterBakery.Java

  - Start with 2 threads:
  - Finished with total of 0 in 2065 ms
  - Start with 4 threads:
  - Finished with total of 0 in 6027 ms
  - Start with 8 threads:
  - Finished with total of 0 in 16092 ms
  - Start with 16 threads:
  - ... Not Finished ... (more than 20 minutes)
- CounterFilter.java
  - Start with 2 threads:
  - Finished with total of 0 in 2146 ms
  - Start with 4 threads:
  - Finished with total of 0 in 10337 ms
  - Start with 8 threads:
  - Finished with total of 0 in 55366 ms
  - Start with 16 threads:
  - ... Not Finished ...
- CounterLock.java
  - Start with 2 threads:
  - Finished with total of 0 in 264 ms
  - Start with 4 threads:
  - Finished with total of 0 in 254 ms
  - Start with 8 threads:
  - Finished with total of 0 in 506 ms
  - Start with 16 threads:
  - Finished with total of 0 in 988 ms
- CounterMonitor.java
  - Start with 2 threads:
  - Finished with total of 0 in 438 ms
  - Start with 4 threads:
  - Finished with total of 0 in 1032 ms
  - Start with 8 threads:
  - Finished with total of 0 in 1936 ms
  - Start with 16 threads:
  - Finished with total of 0 in 3816 ms
- Note: From the different implementations, the worst one are for sure the Filter and Bakery ones. The Counter.java is fast but encounters a lot of conflicts, since it does not implement any lock and synchronized strategies to avoid concurrent access, hence giving a lot of errors and inconsistencies.

2 Task 2: Read/Write implementation

1. Read-write lock:

   A read-write lock allows either a single writer or multiple readers to execute in a critical section. Provide an implementation of a read-write lock in Java. You can use synchronized methods and the wait/notify mechanism if you wish. The class should provide the 4 methods lockRead(), unlockRead(), lockWrite(), and unlockWrite(). This implementation does not need to be FIFO, starvation-free, nor reentrant. HINT: you might want to keep track of the number of readers and writers.

2. Starvation-free read-write lock

   Try to make the read-write lock starvation free for writes (a writer cannot be blocked forever by readers continuously requesting and acquiring the lock).

3. FIFO and reentrant read-write lock (optional):

   Try to make the read-write lock FIFO and reentrant.

   - Answers:
     (a) See the script ReadWrite.java
     (b) See the script ReadWriteAging.java
     (c) See the script ReadWriteReentrant.java
       - Note: This is the optional Exercise of the second part, I didn't have time to finish it before the deadline by finding all bugs and problems that happened while trying to adapt the implementation of ReadWriteAging.java, still wanted to hand in to show my effort on trying it.

3 Task 3: Linearizability and Sequential Consistency

Are the following histories linearizable or sequentially consistent? Explain your answers and write the equivalent linearizable/sequential consistent histories where applicable.

1. Read/write register

   a) Concurrent threads A, B, C, register r:

```
A:  r.write(1)
C:  r.read()
A:  r:void
A:  r.write(2)
C:  r:2
C:  r.read()
B:  r.read()
A:  r:void
C:  r:1
A:  r.write(1)
B:  r:1
A:  r:void
```

   - Answer: please check the file Assignment_2.3.pdf

   b) Concurrent threads A, B, C, register r.

```
A:  r.write(1)
B:  r.read()
A:  r:void
A:  r.write(2)
A:  r:void
A:  r.write(1)
B:  r:1
C:  r.read()
A:  r:void
C:  r:2
```

   - Answer: please check the file Assignment_2.3.pdf

2. Stack:

   We have the following operations:

- push(x) pushes element x on the stack, returns void;
- pop() retrieves an element from the stack;
- empty() returns true if stack is empty and false otherwise.

a) Concurrent threads A, B, C, stack s:

```
C:  s.empty()
A:  s.push(10)
B:  s.pop()
A:  s:void
A:  s.push(20)
B:  s:10
A:  s:void
C:  s:true
```

- Answer: please check the file Assignment_2.3.pdf

b) Concurrent threads A, B, C, stack s:

```
A:  s.push(10)
B:  s.push(10)
A:  s:void
A:  s.pop()
B:  s:void
B:  s.empty()
A:  s:10
B:  s:true
A:  s.pop()
A:  s:10
```

- Answer: please check the file Assignment_2.3.pdf

3. Queue:

   We have the following operations:

   - enq(x) inserts element x in the queue, returns void;
   - deq() retrieves an element from the queue.

   a) Concurrent threads A, B, C, queue q.

```
A:  q.enq(x)
B:  q.enq(y)
A:  q:void
B:  q:void
A:  q.deq()
C:  q.deq()
A:  q:y
C:  q:y
```

   - Answer: please check the file Assignment_2.3.pdf