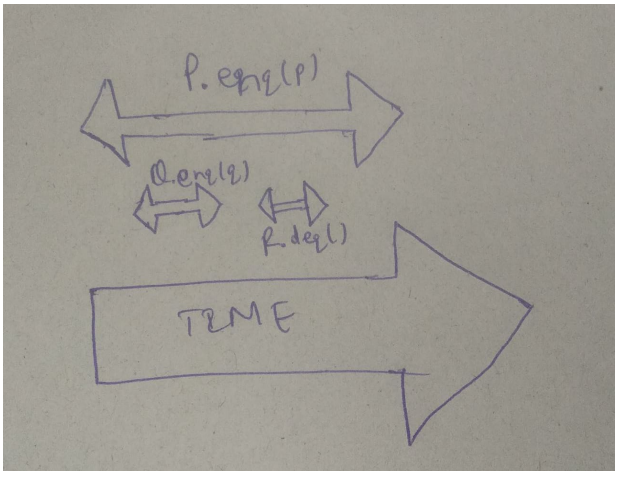
Theory Assignment 1

Aaryan, CO21BTECH11001

Q.1

The AtomicInteger class (in the java.util.concurrent.atomic package) is a container for an integer value. One of its methods is boolean compareAndSet(int expect, int update). This method compares the object's current value with expect. If the values are equal, then it atomically replaces the object's value with update and returns true. Otherwise, it leaves the object's value unchanged, and returns false. This class also provides int get() which returns the object's value. Consider the FIFO queue implementation shown in Fig. 3.13. It stores its items in an array items, which, for simplicity, we assume has unbounded size. It has two AtomicInteger fields: head is the index of the next slot from which to remove an item, and tail is the index of the next slot in which to place an item. Give an example showing that this implementation is not linearizable.

Ans.



In this example, thread P and thread Q are calling the method enq() and thread R is calling the method deq().
Here, thread P will get slot 0 and thread Q will get slot 1. But, thread Q will set the item before thread P.

Therefore, when thread R will call deq(), it will find that slot 0 (head) is a NULL value, therefore it will throw an empty exception.
For the sake of proof, let's assume that there exists a sequential history S which is a linearization of H.
Since enq() precedes deq() in H, therefore it should hold true in S as well.
Now, since deq() is called after enq(), therefore in a sequential queue data structure, it will not throw an empty exception, but dequeues the element at the head of the queue. This contradicts our assumption.

Since there is no legal sequential history of H, therefore this implementation is non linearizable.

Q.2

• Give an execution showing that the linearization point for enq() cannot occur at Line 15. (Hint: Give an execution in which two enq() calls are not linearized in the order they execute Line 15.)

Ans.
There exists an execution where two enq() calls are not linearized in the order they execute Line 15.
Here is an example:

There are two threads A and B which are enqueuing items a and b respectively in the queue.
There is a thread named C which is calling deq() twice.

1. A calls getAndIncrement() at Line 15, which returns 0.
2. B calls getAndIncrement() at Line 15, which returns 1.
3. B stores item b at array index 1.
4. C finds array index 0 empty.
5. C finds array index 1 full, dequeues b.
6. A stores item a at index 0.
7. C again calls deq() and now finds index 0 full and dequeues the item a.

• Give another execution showing that the linearization point for enq() cannot occur at Line 16.

Ans.

There exists an execution where two enq() calls are not linearized in the order they execute Line 16.
Here is an example:

There are two threads A and B which are enqueuing items a and b respectively in the queue.
There is a thread named C which is calling deq() twice.

1. A calls getAndIncrement() at Line 15, which returns 0.
2. B calls getAndIncrement() at Line 15, which returns 1.
3. B stores item b at array index 1 (Line 16).
4. A stores item a at index 0 (Line 16).
5. C finds array index 0 full and dequeues a.
6. C finds array index 1 full and dequeues b.

• Since these are the only two memory accesses in enq(), we must conclude that enq() has no single linearization point. Does this mean enq() is not linearizable?

Ans.
Although these are the only two memory accesses in enq(), these examples do not imply that enq() is not linearizable. It just shows that we cannot define a single linearization point that works for all method calls.