Theory Assignment 1: Linearizability
CO21BTECH11004

Que 3.7)
Sol: -
Consider two threads, T1 and T2.
T1 and T2 calls enq(1) and enq(2), respectively.

- T1 gets slot=0 and increment tail to 1.
- T2 gets slot=1 and increment tail to 2.
- T2 updates the value at index = 1 with value 2.

Thread T2 has completed enq(2) and calls deq().
Till now, T1 has not updated the index.

- T2 gets slot = 0
- At index=0, value = NULL
- EmptyException().

Here enq(2) is completed but in dequeue, EmptyException is thrown.

enq(2) precedes deq(), so the queue can't be empty; still EmptyExceptino is thrown on dequeue. Also, no deq() has occurred, and removed 2 previously.

This inconsistency in the execution shows that this implementation of the FIFO queue is *not linearizable*.

Here, thread T1 didn't update its slot (index 0), The reason for this may be thread suspension or pausing, in which the CPU schedules the threads based on some algorithm.

Using atomic and CAS guarantees proper synchronization of the indices but doesn't guarantee correct sequencing of operations.

Que 3.10) Part 1: -
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.)
Sol: -

Line 15:- *int i = tail.getAndIncrement();*

Take two enq() calls by thread T1 and T2 calls enq(1) and enq(2), line 15:

1: T1 : calls get increment, return 0, i = 0.
2: T2 : calls get increment, return 1, i = 1.

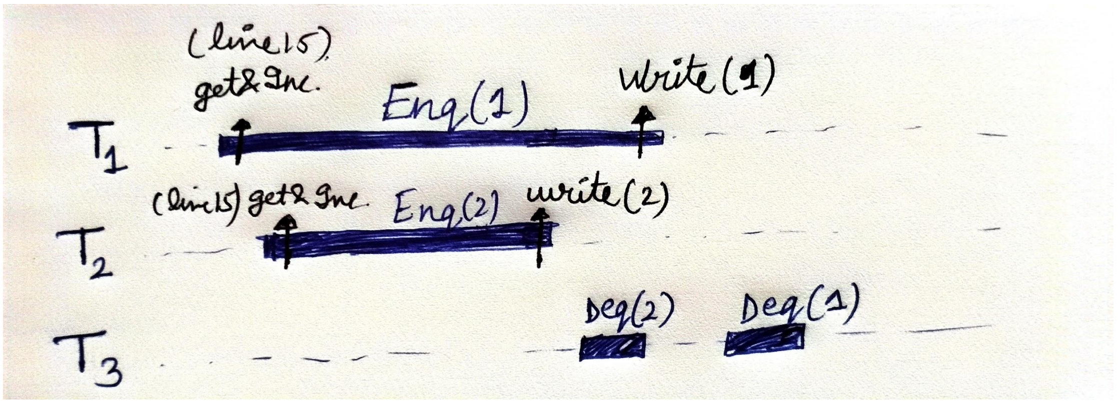
Now T1 and T2 past the linearizable point, not allowed to reorder them to fit.

3: T2: (line16) items[1] = 2
4: T3: check i=0, find null
5: T3: check i=1, find not null, set null and return items[1]
6: T1 : (line 16) items[0] = 1
7: T3: check i=0, find not null, set null and returns items[0]

Here, thread T3 calls deq() two times.

By the order at linearizable point (line 15), enq(1), then enq(2), but in our execution, first deq() returns 2, but the expected was 1.

By this execution, it can be concluded that line 15 can't be a linearizable point.



Part2: -
Give another execution showing that the linearization point for enq() cannot occur at line 16.
Sol: -

Line 16:- *items[i].set(x);*

1: T1 : calls get increment, return 0, i = 0.
2: T2 : calls get increment, return 1, i = 1.

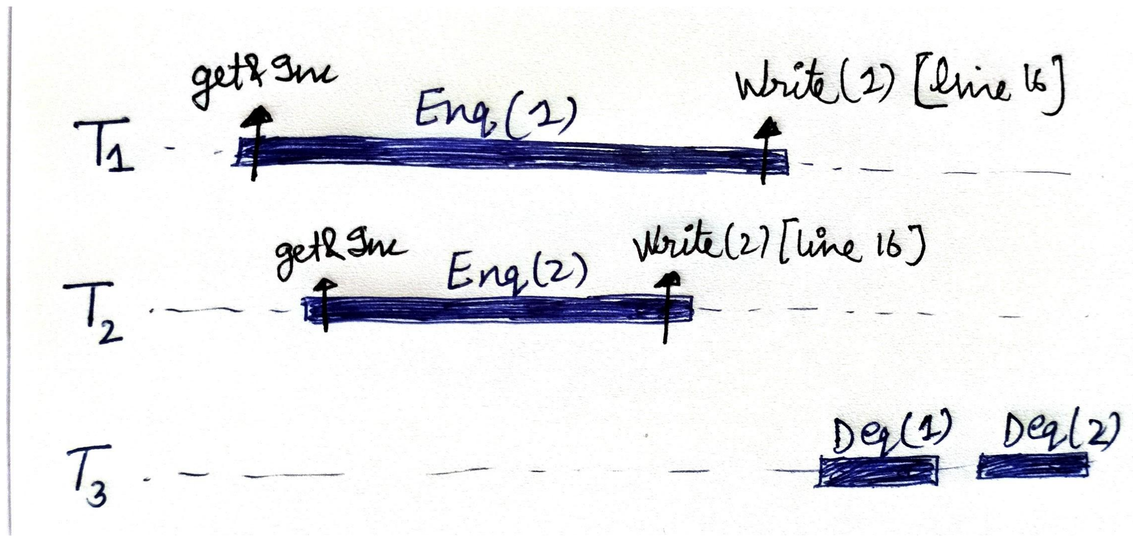
For line 16, writes happen in order T2 than T1.

3: T2: (line16) items[1] = 2
4: T1 : (line 16) items[0] = 1

5: T3: check i=0, find not null, set null and returns items[0]
6: T3: check i=0, find null
7: T3: check i=1, find not null, set null and return items[1]

By the order at the linearizable point (line 16), write items[1] then items[0], but on deq(), items[0] then items[1].

By this execution, it can be concluded that line 16 can't be a linearizable point.



Part3: -
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?
Sol: -
No, it doesn’t mean the enq() method is not linearizable.
We can’t define a linearizable point for lines 15 or 16 that works for all calls for the method enq().