




```

1  <<Start Of Thread 1>>
2  [TestMethod]
3  [HostType("Chess")]
4  [TestProperty("ChessMonitorVolatiles", "True")]
5  [TestProperty("ChessBound", "4")]
6  public void TestLockFreeQueueWithError2() {
7      QueueTest qt = new QueueTest();
8      LockFreeQueue<String> lfq =
9          new LockFreeQueueWithError2<String>();
10     EnqContainer firstParams, secondParams;
11     firstParams.q = secondParams.q = lfq;
12     firstParams.first = "a";
13     firstParams.second = "b";
14     secondParams.first = "c";
15     secondParams.second = "d";
16     Thread t2 = new Thread(
17         new ParameterizedThreadStart(
18             qt.EnqueueTwoStrings));
19     Thread t3 = new Thread(
20         new ParameterizedThreadStart(
21             qt.DequeueThreeStrings));
22     t2.Start(secondParams);
23     t3.Start(lfq);
24     EnqueueTwoStrings(firstParams);
25
26     public void EnqueueTwoStrings (Object o) {
27         EnqContainer e = (EnqContainer)o;
28         e.q.enqueue(e.first);
29         public override void enqueue(T x) {
30             <x == "a">
31             Node<T> node = new Node<T>(x);
32             <1st Iteration of 1>
33             while (true) {
34                 Node<T> last = tail;
35                 Node<T> next = last.next;
36                 <True>
37                 if (last == tail) {
38                     <True>
39                     if (next == null) {
40                         <True>
41                         if (next == Interlocked.CompareExchange(
42                             ref last.next,node,next)) {
43                             <1st Preemption of 4 - goto thread 2>
44                             Interlocked.CompareExchange (
45                                 ref tail, node, last);
46                             return;
47                         }
48                     }
49                 }
50             }
51         }
52     }
53 }

```

KEY

-  Enter into a function call
-  Shows the order of execution.
-  Code that is not executed.

```

54 <<Start Of Thread 2>>
55 public void EnqueueTwoStrings (Object o) {
56     EnqContainer e = (EnqContainer)o;
57     e.q.enqueue(e.first);
58     public override void enqueue(T x) {
59         <x == "c">
60         Node<T> node = new Node<T>(x);
61         <1st Iteration of ?>
62         while (true) {
63             Node<T> last = tail;
64             Node<T> next = last.next;
65             <True>
66             if (last == tail) {
67                 <False - Does not Execute>
68                 if (next == null) {
69                     ...
70                 }
71                 <Does Execute>
72             } else {
73                 // The mistake was introduced by
74                 // replacing an atomic operation
75                 // (i.e. CompareExchange) with a
76                 // non-atomic operation.
77                 <True>
78                 if (tail == last)
79                     <2nd Preemption of 4 - goto thread 1>
80                     tail = next;
81             }
82         }
83     }
84 }

```

```

85 <<Start Of Thread 3>>
86 public void DequeueThreeStrings(Object o) {
87     LockFreeQueue<String> lfq =
88         (LockFreeQueue<String>)o;
89     try { lfq.dequeue(); } catch (EmptyException e) {}
90     public virtual T dequeue() {
91         <1st Iteration of 1>
92         while (true) {
93             Node<T> first = head;
94             Node<T> last = tail;
95             Node<T> next = first.next;
96             <True>
97             if (first == head) {
98                 <False - Does not Execute>
99                 if (first == last) {
100                     ...
101                 }
102                 <Does Execute>
103             } else {
104                 T val = next.val;
105                 <True>
106                 if (first == Interlocked.CompareExchange(
107                     ref head,next,first))
108                     <val == "a">
109                     return val;
110             }
111         }
112     }
113 }

```

47

13

9

97
98
99
100
101

<pre> tail = next; } } } }</pre>	<pre> return val; } } } }</pre>
<<End Of Thread 2>>	<<End Of Thread 3>>