javaf / array-queue

Array queue is a bounded lock-based FIFO queue using an array. It uses 2 separate locks for head and tail.

repl.it/@wolfram77/array-queue#readme.md

MIT License

0 stars 0 forks

Star Unwatch

Code Issues Pull requests Actions Projects Wiki Security

master

wolfram77 wrote README w/ output ... 7 days ago 5

View code

README.md

Array queue is a bounded lock-based FIFO queue using an array. It uses 2 separate locks for head and tail.

```
enq():
1. Lock tail.
2. Try enq.
3. Unlock tail.


deq():
1. Lock head.
2. Try deq.
3. Unlock head.


tryEnq():
1. Ensure queue is not full
2. Save data at tail.
3. Increment tail.
```

```
tryDeq():
1. Ensure queue is not empty.
2. Return data at head.
3. Increment head.



## OUTPUT
Starting 10 threads with sequential queue
2: failed enq
5: failed deq
6: failed deq
7: failed deq
8: failed deq
1: failed deq
4: failed deq
9: failed deq
1: dequeued 0/1000 values
2: dequeued 0/1000 values
4: dequeued 0/1000 values
5: dequeued 698/1000 values
6: dequeued 0/1000 values
7: dequeued 0/1000 values
8: dequeued 0/1000 values
9: dequeued 0/1000 values
Was LIFO? false

Starting 10 threads with array queue
Was LIFO? true
```

See ArrayQueue.java for code, Main.java for test, and repl.it for output.

references

- The Art of Multiprocessor Programming :: Maurice Herlihy, Nir Shavit

Languages

- **Java** 100.0%

73 lines (66 sloc) 1.67 KB

```
1  import java.nio.*;
2  import java.util.concurrent.locks.*;
3
4  // Array queue is a bounded lock-based FIFO queue using
5  // an array. It uses 2 separate locks for head and tail.
6
7  class ArrayQueue<T> {
8    Lock headLock;
9    Lock tailLock;
10   T[] data;
11   int head;
12   int tail;
13   // headLock: lock for enq() at head
14   // tailLock: lock for deq() at tail
15   // data: array of values in stack
16   // head: front of queue (0)
17   // tail: rear of queue (0)
18
19   @SuppressWarnings("unchecked")
20   public ArrayQueue(int capacity) {
21     headLock = new ReentrantLock();
22     tailLock = new ReentrantLock();
23     data = (T[]) new Object[capacity+1];
24     head = 0;
25     tail = 0;
26   }
27
```

```
28    // 1. Lock tail.
29    // 2. Try enq.
30    // 3. Unlock tail.
31    public void enq(T x) {
32      try {
33      tailLock.lock();   // 1
34      tryEnq(x);         // 2
35      } finally {
36      tailLock.unlock(); // 3
37      }
38    }
39
40    // 1. Lock head.
41    // 2. Try deq.
42    // 3. Unlock head.
43    public T deq() {
44      try {
45      headLock.lock();   // 1
46      return tryDeq();   // 2
47      } finally {
48      headLock.unlock(); // 3
49      }
50    }
51
52    // 1. Ensure queue is not full
53    // 2. Save data at tail.
54    // 3. Increment tail.
55    protected void tryEnq(T x) {
56      int tail2 = (tail + 1) % data.length;  // 1, 3
57      if (tail2 == head)                     // 1
58        throw new BufferOverflowException(); // 1
59      data[tail] = x; // 2
60      tail = tail2;   // 3
61    }
62
63    // 1. Ensure queue is not empty.
64    // 2. Return data at head.
65    // 3. Increment head.
66    protected T tryDeq() {
67      if (head == tail)                      // 1
68        throw new BufferUnderflowException(); // 1
69      T x = data[head];                 // 2
70      head = (head + 1) % data.length; // 3
71      return x;                         // 2
72    }
73  }
```

javaf / array-queue

Code Issues Pull requests Actions Projects Wiki Security

master

array-queue / Main.java / Jump to

wolfram77 wrote README w/ output History

1 contributor

Raw Blame

102 lines (95 sloc) 2.69 KB

```
1  import java.util.*;
2
3  class Main {
4    static Deque<Integer> queue;
5    static ArrayQueue<Integer> concurrentQueue;
6    static List<Integer>[] deqValues;
7    static int TH = 10, NUM = 1000;
8
9    // Each unsafe thread enqs N numbers and deqs N, adding
10   // them to its own deqValues for checking; using Java's
11   // sequential queue implementation, ArrayDeque.
12   static Thread unsafe(int id, int x, int N) {
13     return new Thread(() -> {
14       String action = "enq";
15       try {
16       for (int i=0, y=x; i<N; i++)
17         queue.addLast(y++);
18       Thread.sleep(1000);
19       action = "deq";
20       for (int i=0; i<N; i++)
21         deqValues[id].add(queue.removeFirst());
22       }
23       catch (Exception e) { log(id+": failed "+action); }
24     });
25   }
26
27   // Each safe thread enqs N numbers and deqs N, adding
```

```
28    // them to its own deqValues for checking; using
29    // ArrayQueue.
30    static Thread safe(int id, int x, int N) {
31      return new Thread(() -> {
32        String action = "enq";
33        try {
34        for (int i=0, y=x; i<N; i++)
35          concurrentQueue.enq(y++);
36        Thread.sleep(1000);
37        action = "deq";
38        for (int i=0; i<N; i++)
39          deqValues[id].add(concurrentQueue.deq());
40        }
41        catch (Exception e) { log(id+": failed "+action);
42        e.printStackTrace(); }
43      });
44    }
45
46    // Checks if each thread dequeued N values, and they are
47    // globally unique.
48    static boolean wasLIFO(int N) {
49      Set<Integer> set = new HashSet<>();
50      boolean passed = true;
51      for (int i=0; i<TH; i++) {
52        int n = deqValues[i].size();
53        if (n != N) {
54          log(i+": dequeued "+n+"/"+N+" values");
55          passed = false;
56        }
57        for (Integer x : deqValues[i])
58          if (set.contains(x)) {
59            log(i+": has duplicate value "+x);
60            passed = false;
61          }
62        set.addAll(deqValues[i]);
63      }
64      return passed;
65    }
66
67    @SuppressWarnings("unchecked")
68    static void testThreads(boolean safe) {
69      queue = new ArrayDeque<>();
70      concurrentQueue = new ArrayQueue<>(TH*NUM);
71      deqValues = new List[TH];
72      for (int i=0; i<TH; i++)
73        deqValues[i] = new ArrayList<>();
74      Thread[] threads = new Thread[TH];
75      for (int i=0; i<TH; i++) {
```

```
76        threads[i] = safe?
77          safe(i, i*NUM, NUM) :
78          unsafe(i, i*NUM, NUM);
79        threads[i].start();
80      }
81      try {
82      for (int i=0; i<TH; i++)
83        threads[i].join();
84      }
85      catch (Exception e) {}
86    }
87
88    public static void main(String[] args) {
89      log("Starting "+TH+" threads with sequential queue");
90      testThreads(false);
91      log("Was LIFO? "+wasLIFO(NUM));
92      log("");
93      log("Starting "+TH+" threads with array queue");
94      testThreads(true);
95      log("Was LIFO? "+wasLIFO(NUM));
96      log("");
97    }
98
99    static void log(String x) {
100     System.out.println(x);
101   }
102 }
```