Subhajit Sahu
2018801013

28.10.2020

TUTORIAL-2

126. design an unbounded lock-based Stack<T> implementation based on linked list.

A.
```
class Stack<T> {
  Node top = null;
  Lock lock = new ReentrantLock();

  public void push(T x) {
    Node n = new Node(x);
    lock.lock();
    n.next = top;
    top = n;
    lock.unlock();
  }

  public T pop() throws EmptyException {
  try { lock.lock();
      ~~if~~ Node n = top;
      if (n == null) throw new EmptyException();
      top = n.next;
      return n.value;
    } finally {
      lock.unlock();
    }
  }
} // Stack<T>
```

Q.7 design a bounded lock-based Stack<T> using an array:
1. use a single lock and a bounded array.
2. try to make your algorithm lock-free. where do you run into difficulty?

A. 1.

```
class Stack<T> {
    T[] data;
    int top;
    Lock lock;

    public Stack (int capacity) {
        data = ~~new~~ (T[]) new Object[capacity];
        top = 0;
        lock = new ReentrantLock();
    }

    public void push (T x) throws BufferOverflowExcep {
        ~~Stacke~~ try {
        lock.lock();
        if (top == data.length-1)
            throw new BufferOverflowException();
        data [top] = x;
        top = top+1;
        } finally {
          lock.unlock();
        }
    }
public T pop() throws BufferUnderflowException {
    try {
        lock.lock();
        if (top==0) throw new BufferUnderflowException();
        top = top-1;
        return data[top];
    } finally {
        lock.unlock();
    }
  }
} // Stack <T>
```

2.

```
push (x):                      pop():
(1) cas (top, top+1);          (2) cas (top, top-1);
(4) a[oldTop] = x;             (3) return a[top];
```

consider above algorithm using CAS for push & pop (excluding the bound checks). given (1) (2) (3) (4) is an example execution order, where push only manages to execute the CAS after which a pop completes (2), (3) and finally push completes (4). so this pop() managed to execute when push() didnt have a chance to actually store the pushed value. hence, pop() would read a garbage value here. this is a problem that can occurer with a bounded array-based lock-free stack, because set value and modify top are not atomic.

javaf / **array-stack**

Array stack is a bounded lock-based stack using an array. It uses a common lock for both push and pop operations.

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

MIT License

0 stars 0 forks

Star Unwatch

Code Issues Pull requests Actions Projects Wiki Security

master

wolfram77 added output in README … 8 days ago 4

View code

README.md

Array stack is a bounded lock-based stack using an array. It uses a common lock for both push and pop operations.

```
push():
1. Lock stack.
2. Try push.
3. Unlock stack.
```

```
pop():
1. Lock stack.
2. Try pop.
3. Unlock stack.
```

```
tryPush():
1. Ensure stack is not full
2. Save data at top.
3. Increment top.
```

```
tryPop():
1. Ensure stack is not empty.
2. Decrement top.
3. Return data at top.



## OUTPUT
Starting 10 threads with sequential stack
7: failed pop
1: failed pop
5: failed pop
8: failed pop
9: failed pop
1: popped 0/1000 values
5: popped 158/1000 values
7: popped 0/1000 values
8: popped 0/1000 values
9: popped 31/1000 values
Was LIFO? false

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

See ArrayStack.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%

javaf / array-stack

Code Issues Pull requests Actions Projects Wiki Security

master

array-stack / ArrayStack.java / Jump to

wolfram77 added output in README History

1 contributor

Raw Blame

66 lines (59 sloc) 1.48 KB

```java
import java.nio.*;
import java.util.concurrent.locks.*;

// Array stack is a bounded lock-based stack using an
// array. It uses a common lock for both push and pop
// operations.

class ArrayStack<T> {
  Lock lock;
  T[] data;
  int top;
  // lock: common lock for push, pop
  // data: array of values in stack
  // top: top of stack (0 if empty)

  @SuppressWarnings("unchecked")
  public ArrayStack(int capacity) {
    lock = new ReentrantLock();
    data = (T[]) new Object[capacity];
    top = 0;
  }

  // 1. Lock stack.
  // 2. Try push.
  // 3. Unlock stack.
  public void push(T x) throws BufferOverflowException {
    try {
```

```
    lock.lock();   // 1
    tryPush(x);    // 2
    } finally {
    lock.unlock(); // 3
    }
  }

  // 1. Lock stack.
  // 2. Try pop.
  // 3. Unlock stack.
  public T pop() throws BufferUnderflowException {
    try {
    lock.lock();     // 1
    return tryPop(); // 2
    } finally {
    lock.unlock();   // 3
    }
  }

  // 1. Ensure stack is not full
  // 2. Save data at top.
  // 3. Increment top.
  protected void tryPush(T x)
    throws BufferOverflowException {
    if (top == data.length)                // 1
      throw new BufferOverflowException(); // 1
    data[top++] = x; // 2, 3
  }

  // 1. Ensure stack is not empty.
  // 2. Decrement top.
  // 3. Return data at top.
  protected T tryPop()
    throws BufferUnderflowException {
    if (top == 0)                           // 1
      throw new BufferUnderflowException(); // 1
    return data[--top]; // 2, 3
  }
}
```

javaf / array-stack

Code Issues Pull requests Actions Projects Wiki Security

master

array-stack / Main.java / Jump to

wolfram77 added output in README History

1 contributor

Raw Blame

102 lines (95 sloc) 2.71 KB

```java
import java.util.*;

class Main {
  static Deque<Integer> stack;
  static ArrayStack<Integer> concurrentStack;
  static List<Integer>[] poppedValues;
  static int TH = 10, NUM = 1000;

  // Each unsafe thread pushes N numbers and pops N, adding
  // them to its own poppedValues for checking; using Java's
  // sequential stack implementation, ArrayDeque.
  static Thread unsafe(int id, int x, int N) {
    return new Thread(() -> {
      String action = "push";
      try {
      for (int i=0, y=x; i<N; i++)
        stack.push(y++);
      Thread.sleep(1000);
      action = "pop";
      for (int i=0; i<N; i++)
        poppedValues[id].add(stack.pop());
      }
      catch (Exception e) { log(id+": failed "+action); }
    });
  }

  // Each safe thread pushes N numbers and pops N, adding
  // them to its own poppedValues for checking; using
  // ArrayStack.
  static Thread safe(int id, int x, int N) {
    return new Thread(() -> {
      String action = "push";
      try {
      for (int i=0, y=x; i<N; i++)
        concurrentStack.push(y++);
      Thread.sleep(1000);
      action = "pop";
      for (int i=0; i<N; i++)
        poppedValues[id].add(concurrentStack.pop());
      }
      catch (Exception e) { log(id+": failed "+action);
      e.printStackTrace(); }
    });
  }

  // Checks if each thread popped N values, and they are
  // globally unique.
  static boolean wasLIFO(int N) {
    Set<Integer> set = new HashSet<>();
    boolean passed = true;
    for (int i=0; i<TH; i++) {
      int n = poppedValues[i].size();
      if (n != N) {
        log(i+": popped "+n+"/"+N+" values");
        passed = false;
      }
      for (Integer x : poppedValues[i])
        if (set.contains(x)) {
          log(i+": has duplicate value "+x);
          passed = false;
        }
      set.addAll(poppedValues[i]);
    }
    return passed;
  }

  @SuppressWarnings("unchecked")
  static void testThreads(boolean safe) {
    stack = new ArrayDeque<>();
    concurrentStack = new ArrayStack<>(TH*NUM);
    poppedValues = new List[TH];
    for (int i=0; i<TH; i++)
      poppedValues[i] = new ArrayList<>();
    Thread[] threads = new Thread[TH];
    for (int i=0; i<TH; i++) {
      threads[i] = safe?
        safe(i, i*NUM, NUM) :
        unsafe(i, i*NUM, NUM);
      threads[i].start();
    }
    try {
    for (int i=0; i<TH; i++)
      threads[i].join();
    }
    catch (Exception e) {}
  }

  public static void main(String[] args) {
    log("Starting "+TH+" threads with sequential stack");
    testThreads(false);
    log("Was LIFO? "+wasLIFO(NUM));
    log("");
    log("Starting "+TH+" threads with array stack");
    testThreads(true);
    log("Was LIFO? "+wasLIFO(NUM));
    log("");
  }

  static void log(String x) {
    System.out.println(x);
  }
}
```