Lucas Rosa

04/23/16

2847275

Homework Assignment 3

Chapter 10

Exercise 121

The problem says to design a lock-based array implementation of a queue. I decided to use go and this is what the head/tail and queue structs would look like:

```
// head & tail pointers
type sentinel struct {
  index int
  lock sync.Mutex
}

// queue
type Queue struct {
  head, tail *sentinel
  elements [CAPACITY]interface{}
  size uint64
}
```

Step one says to allow for parallelism by using two separate locks for head and tail. Now for add and remove methods:

```
func (q *Queue) enqueue(v interface{})  {
  q.tail.Lock()

  q.elements[q.tail.index] = v

  q.tail.index = (q.tail.index+1)%CAPACITY

  q.tail.Unlock()
}

func (q *Queue) dequeue() (v interface{}) {
  q.head.Lock()

  q.elements[q.head.index] = nil

  q.head.index = (q.head.index+1)%CAPACITY

  q.head.Unlock()
}
```

There were issues transforming this into a lock free implementation. Other than having issues translating some of the Java code from the book to go-lang, the bounded array aspect of this was difficult to deal with. When dealing with a bounded structure slow downs can be expected especially in the case of a full structure. More efficient implementation use some kind of waiting mechanism to deal with this but you still have to wait. Also it would be easier to implement this if a linked list was used with pointers to nodes.

Exercise 125

1. In this example `enq()` is wait free because it will not block any threads from progressing with it's execution, all threads are guaranteed to complete within a known amount of steps. Since it is wait free it is also lock-free. The `deq()` method is not wait-free because any particular thread is not necessarily going to complete in

a known amount of steps. This is caused by the `while` loop which may loop forever given the "proper" conditions. Although it may not be wait-free, it is lock-free. One thread stuck in an infinite or lengthy loop will not stop system-wide progress. Other threads can perhaps `deq()` and definitely `enq()`. Also one can naively say that the `deq()` method doesn't use a lock. Lock freedom is more about guaranteeing system-wide progress than it is about not using locks in the data structure.

2.

Chapter 11

Exercise 131

Chapter 13

Exercise 159

Exercise 160

Chapter 14

Exercise 163

Exercise 167

Exercise 168

Exercise 172