

---

**Lecture: Concurrency Theory and Practise**

<http://proglang.informatik.uni-freiburg.de/teaching/concurrency/2014ws/>

---

**Exercise Sheet 6**

2015-01-23

**I. Theory****I.1. Blocking Single Enqueuer and Single Dequeuer Queue**

Consider the following simple lock-free queue for a single enqueuer and a single dequeuer.

```

1 class TwoThreadLockFreeQueue<T> {
2   int head = 0, tail = 0;
3   T[] items;
4   public TwoThreadLockFreeQueue(int capacity) {
5     head = 0;
6     tail = 0;
7     item = (T[]) new Object[capacity];
8   }
9
10  public void enq(T x) {
11    while(tail - head == items.length) {}
12    items[tail % items.length] = x;
13    tail++;
14  }
15
16  public T deq() {
17    while (tail - head == 0) {}
18    T x = items[head % items.length];
19    head++;
20    return x;
21  }
22 }
```

This queue is blocking, that is, removing an item from an empty queue or inserting an item to a full one causes the threads to block (spin). The surprising thing about this queue is that it requires only loads and stores and not a more powerful read-modify-write synchronization operation. Does it however require the use of a memory barrier? If not, explain, and if so, where in the code is such a barrier needed and why.

**I.2. Fine-grained linked lists**

Explain why the fine-grained locking algorithm for linked lists is not subject to deadlock.

**I.3. Linearizable lock-free linked list**

Explain why the following cannot happen in the LockFreeList algorithm. A node with item  $x$  is logically but not yet physically removed by some thread, then the same item  $x$  is added into the list by another thread, and finally a `contains()` call by a third thread traverses the list, finding the logically removed node, and returning *false*, even though the linearization order of the `remove()` and `add()` calls implies that  $x$  is in the set.

**I.4. Bounded DEQueue**

1. In the `popBottom()` method of the `BDEQueue` from the slides, the `bottom` field is volatile to assure that in `popBottom()` the decrement at Line 3 is immediately visible. Describe a scenario that explains what could go wrong if `bottom` were not declared as volatile.
2. Why should we attempt to reset the `bottom` field to zero as early as possible in the `popBottom()` method? Which point is the earliest at which this reset can be done safely? Can our `BDEQueue` overflow anyway? Describe how.

## II. Practice

### II.1. Pool party!

Creating a new thread for each task in a program can lead to major performance issues as thread spanning is expensive. Thread pools limit the number of threads and allow thus a good capacity utilization.

Fill an array of size  $n$  (random value between 0 and 100,000) with random numbers from 0 to  $n$ . For each number, count its frequency in the list. To increase the application's performance, the counting tasks are to be done in different threads.

Implement a thread factory for the counting threads to be used with Java's `ThreadPoolExecutor`. Further, extend the `ThreadPoolExecutor` with statistics about the total execution time, number of tasks done, average execution time of a task, ...

### II.2. Simple Numerics

Implement a class for vectors of size  $n$  (great  $n$ , e.g.  $n > 10000$ ),  $\vec{x} = (x_1, \dots, x_n)$  with  $x_i \in \mathbb{R}$ , as they are used in numerical computations. It should provide parallel implementations of the following methods:

- summation of all vector entries:  $sum(\vec{x}) = x_1 + \dots + x_n$
- addition of two vectors:  $add(\vec{x}, \vec{y}) = (x_1 + y_1, \dots, x_n + y_n)$
- scalar multiplication:  $scal(a, \vec{x}) = (ax_1, \dots, ax_n)$
- length of a vector:  $length(\vec{x}) = \sqrt{x_1^2 + \dots + x_n^2}$
- dot product of two vectors:  $prod(\vec{x}, \vec{y}) = x_1y_1 + \dots + x_ny_n$

For simplicity, you may assume that  $n = 2^k$  for some  $k \in \mathbb{N}$ .

---

### Submission

- Deadline: **2015-02-05, 23:59**
- Submit theory exercises in PDF format via email to `concurrency@informatik.uni-freiburg.de`. Please name your single file with the scheme: `ex6-name(s).pdf`.
- Submit practical exercises as executable jar-files for each exercise. The file name should include the name of the exercise and your name (example: `philosophers-fennell.jar`). Make sure that you include all source files and libraries you use. Sources should always be documented!
- Late submissions may not be corrected.
- Do not forget to write your name(s) on the exercise sheet.
- You may submit in groups up to 2 people.