Prof. P. Thiemann, L. Fennell, M. Geffken Winter Term 2014/15

Lecture: Concurrency Theory and Practise

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

Exercise Sheet 3

2014-11-28

I. Theory

(There is no practice part this time)

I.1. Queues

Consider this unbounded queue implementation:

```
public class HWQueue<T> {
  AtomicReference<T>[] items;
  AtomicInteger tail ;
  ...
  public void enq(T x) {
    int i = tail.getAndIncrement();
    items[i].set(x);
  }
  public T deq() {
    while (true) {
      int range = tail.get();
      for (int i = 0; i < range; i++) {
        T value = items[i].getAndSet(null);
        if (value != null) {
          return value;
        }
      }
    }
  }
}
```

This queue is blocking, meaning that the deq() method does not return until it has found an item to dequeue. The queue has two fields: items is a very large array (you may assume that we never address an element beyond the array bounds), and tail is the index of the next unused element in the array.

1. Assuming the queue is never empty, are the enq() and deq() methods wait-free? If not, are they lock-free? Explain.
2. Identify the linearization points for enq() and deq(). (Careful! They may be execution-dependent.)

I.2. Progress Conditions

Is the following property equivalent to saying that object x is wait-free?

> For every infinite history H of x, every thread that takes an infinite number of steps in H completes an infinite number of method calls.

Is the following property equivalent to saying that object x is lock-free?

> For every infinite history H of x, an infinite number of method calls are completed.

Justify your answers.

I.3. Memory models

Consider the following Java code snippet.

```java
class Example{
  int x = 0;
  boolean b = false;
  public void writer() {
    x = 42;
    b = true;
  }
  public void reader() {
    if (b == true) {
      int r = 100/x;
    }
  }
}
```

How is it possible that the above code fails with a divide-by-zero error? Fix the code such that this error is impossible.

Optional: Consult the specification of the Java Memory Model[1] to find alternative ways to fix the code.

I.4. "Dependent" Consensus

One can implement a consensus object using read-write registers by implementing a deadlock- or starvation-free mutual exclusion lock. However, this implementation provides only dependent progress, and the operating system must make sure that threads do not get stuck in the critical section so that computation as a whole progresses.

- Implement a consensus object using locks.
- Is the same true for obstruction-freedom, the non-blocking dependent progress condition? Show an obstruction-free implementation of a consensus object using only atomic registers.
- What is the role of the operating system in the obstruction-free solution to consensus? Explain where the critical-state-based proof of the impossibility of consensus breaks down if we repeatedly allow an oracle to halt threads so as to allow others to make progress. (Hint: think of how you could restrict the set of allowed executions)

I.5. Minimal `compareAndSet`

Consider a class of registers MinimalCAS that has a single method compareAndSet(), behaving as discussed in the lecture, but has no method get(). Show that MinimalCAS registers have consensus number ∞.

I.6. `NewCompareAndSet`

Strictly speaking, compareAndSet() is not a real read-modify-write method, as it returns a boolean and not the value that was present before applying the modification function. Use the methods compareAndSet() and get() to implement a newCompareAndSet() method that is a proper read-modify-write method.

Submission

- Deadline: **2014-12-11, 23:59**
- Submit theory exercises in PDF format via email to `concurrency AT informatik.uni-freiburg.de`. Please name your single file with the scheme: `ex3-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!

[1] `http://docs.oracle.com/javase/specs/jls/se7/html/jls-17.html#jls-17.4`

- 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.