

EXAMINATION

Formal Methods and Tools
Faculty of EEMCS
University of Twente

PROGRAMMING PARADIGMS CONCURRENT PROGRAMMING

code: 201400537
date: 5 July 2017
time: 13.45–16.45

- This is an 'open book' examination. You are allowed to have a copy of *Java Concurrency in Practice*, an (unannotated) copy of the course manual, a copy of the (unannotated) lecture (hoorcollege) *slides*, and print outs of the following additional material:
 - A gentle introduction to OpenCL - DrDobbs, by M. Scarpino.
 - B. Chapman, G. Jost and R. van der Pas. Using OpenMP - The Book.
 - Simon Peyton Jones and Satnam Singh. A Tutorial on Parallel and Concurrent Programming in Haskell. Advanced Functional Programming Summer School.
 - The Rust Programming Language, second edition.

You are *not* allowed to take personal notes, solutions to the exercises, and (answers to) previous examinations with you.

- You can earn 100 points with the following **5 questions**. The final grade is computed as the number of points, divided by 10. Students in the *Programming Paradigms* module need to obtain at least a 5.0 for the test. The bonus points that were obtained by participating in the quiz during the tutorial sessions and the first lecture will be added to the final result.

GOOD LUCK!

Question 1 (25 points)

One popular synchronization construct is a barrier. A barrier for a group of threads ensures that all threads stop at this point, and cannot proceed until all other threads have reached this barrier. In this exercise, we look at different mechanisms to implement barriers.

A barrier implementation should support the following operations:

- initialise the barrier, setting the number of threads that will be involved in the synchronisation; and
 - `await`: a thread that calls this operation blocks until all threads involved in the synchronisation have reached the barrier.
- a. (15 pnts.) Implement a barrier in Java **using a count down latch**. Make sure that your barrier is reusable, i.e., it can be reused multiple times to synchronise a group of threads.
- b. (10 pnts.) Use transactional variables to give a Haskell implementation of a barrier.

Question 2 (35 points)

Consider the class `Delivery`.

```
class Delivery {  
  
    private int[] box;  
  
    public Delivery(int n) {  
        box = new int[n];  
        for (int i = 0; i < n; i++) {  
            box[i] = 0;  
        }  
    }  
  
    public void deliver(int parcels, int i) {  
        box[i] = box[i] + parcels;  
    }  
  
    public int empty(int i) {  
        int res = box[i];  
        box[i] = 0;  
        return res;  
    }  
}
```

This implements a sequence of message boxes. Delivery men can add a number of parcels to a box. The owner of the box can empty the box. For simplicity, we only record the number of items in each box.

- a. (6 *pts.*) Make a thread safe version of class `Delivery` in such a way that delivery men that want to deliver parcels in disjoint boxes simultaneously do not block each other.
- b. (2 *pts.*) Specify a suitable postcondition for method `deliver` in your thread safe version of `Delivery`. Explain your answer.
- c. (5 *pts.*) Suppose we want to add a transfer function, which transfer parcels from one box to another. Discuss what the main concurrency-related risk is, when adding this to your thread safe version, and how this risk can be avoided.
- d. (4 *pts.*) Naturally, parcels should not be lost. What sort of fairness is needed to ensure that every parcel will eventually be delivered? Motivate your answer.
- e. (10 *pts.*) Give a thread safe lock-free version of `Delivery`.
- f. (8 *pts.*) Give a message passing implementation for the `deliver` operation for a **single** delivery box. When a delivery man arrives, he tries to send a parcel to the box, and continues trying this, until he succeeds. As multiple delivery men can enter parcels in the box, you have to make sure that the message to confirm the receipt of the parcel is sent to the right delivery man. (It is recommended to give an answer in Rust. However, points will not be subtracted for the concrete syntax, as long as the message passing schema is clear.)

Question 3 (15 points)

- a. (3 pts.) Consider the following fragment of a light handling system.

```
// nrLights is length of lights array
// DIMMODE is some boolean condition

# pragma omp parallel for shared (lights, MAX, DIMMODE) private (i)
for (i = 0; i < nrLights; ++i) {
    lights[i] = MAX;
    if (DIMMODE && ((i % 2) == 0)) {
        lights[i] = MAX/2;
    }
}
```

Describe what will be the effect of the OpenMP pragma on the behaviour of this fragment.

- b. (4 pts.) Consider the following fragment of a light handling system.

```
// nrLights is length of lights array
// length of init is the same as length of lights

# pragma omp parallel for shared (lights, MAX, DIMMODE) private (i)
for (i = 0; i < nrLights; ++i) {
    lights[i] = init[i];
    if (lights[i] < lights[i - 1])
        lights[i] = lights[i - 1];
}
}
```

Describe what will be the effect of the OpenMP pragma on the behaviour of this fragment.

- c. (4 pts.) Give an OpenCL kernel that has the same effect as the sequential version of this second fragment.
- d. (4 pts.) Consider the following OpenCL kernel.

```
//size is length of lights and init array
2
__kernel lights (__global float* lights, __global float* init, int size) {
4     int index = get_global_id(0);
    lights[index] = 0;
6     for (int i = 0; i < size; i++) {
        if (i % (index + 1) == 0) {
8         lights[index] = lights[index] + init[i];
        }
10    }
}
```

Would adding a barrier before the `for`-loop in line 6 change the final value of the `lights` array?

Question 4 (15 points)

- a. (3 pts.) Suppose we transform the following program fragment:

```

l.lock();
try {
    x = 3;
    y = 4;
}
finally {
    l.unlock();
}

```

into the following program fragment

```

l.lock();
try {
    y = 4;
}
finally {
    l.unlock();
}
x = 3;

```

Suppose this is used in a context with other threads that need up-to-date values of variables x and y . Explain whether this program transformation is okay. Motivate your answer.

- b. (3 pts.) Suppose we have the following 3 threads, where initially
- $x = 0$
- ,
- $y = 0$
- , and
- $z = 0$
- , and
- r_1
- ,
- r_2
- and
- r_3
- are local variables.

Thread 1	Thread 2	Thread 3
$x = 10;$	$y = 8;$	$z = 6;$
$\text{int } r_1 = z;$	$\text{int } r_2 = x;$	$\text{int } r_3 = y;$

Give all possible combinations of the final values of r_1 , r_2 and r_3 under a relaxed memory model.

- c. (3 pts.) Consider the following program.

```

class Hotel {
    volatile String[] rooms = new String[34];

    /*@ requires 0 <= number && number < 34;
    void checkIn(Guest name, int number) {
        rooms[number] = name;
    }

    /*@ requires 0 <= number && number < 34;
    void checkOut(int number) {
        rooms[number] = null;
    }
}

```

When a single instance of this class is used by multiple threads, explain whether this program has data races. Motivate your answer.

- d. (6 pts.) Suppose two neighbours, Bob and John, share a garden. Bob has a cat, and John has a dog. Both animals like to be outside in the garden, but they cannot be outside at the same time, as they will start fighting. They agree to use a flag system that signals when their animals are outside, in order to avoid fights. An attempt to implement this system is made in the classes
- `Garden`
- ,
- `Neighbour`
- , and
- `AnimalControl`
- (see Figure 1).

Discuss at least two reasons why this implementation will not ensure that the cat and the dog are outside simultaneously.

```
public class Garden {

    public static final int EMPTY = 0;
    public static final int CAT = 1;
    public static final int DOG = 2;
    private boolean flag = false;

    /*@ invariant animal == EMPTY || animal == CAT || animal == DOG;
    private int animal = EMPTY;

    /*@ requires an == CAT || an == DOG;
    public void enter(int an) {
        animal = an;
    }

    /*@ requires animal == CAT || animal == DOG;
    public void leave() {
        animal = EMPTY;
    }

    public void raiseFlag() {
        flag = true;
    }

    public void lowerFlag() {
        flag = false;
    }

    public boolean inspectFlag() {
        return flag;
    }
}

public class Neighbour extends Thread {

    private final int animal;
    private Garden garden;

    public Neighbour(int an, Garden g) {
        animal = an;
        garden = g;
    }

    public void run () {
        while (garden.inspectFlag());
        garden.raiseFlag();
        garden.enter(animal);
        garden.leave();
        garden.lowerFlag();
    }
}

public class AnimalControl {

    public static void main(String [] args) {
        Garden g = new Garden();
        Neighbour bob = new Neighbour(Garden.CAT, g);
        Neighbour john = new Neighbour(Garden.DOG, g);
        bob.start();
        john.start();
    }
}
```

Figure 1: The Animal Control System

Question 5 (10 points)

Consider the following Rust program:

```

use std::thread;
2 use std::sync::Mutex;
use std::sync::Arc;
4
fn apply(x : usize) -> usize {
6   // apply computation to this job
   }
8
fn main () {
10
12   let n = // number of jobs
   let mut jobs = Vec::new();
14   for i in 0 .. n {
       // initialise jobs
16   }
   let jobs_copy = jobs.clone();
18
   let results = Arc::new(Mutex::new(vec![0;n]));
20   let results_copy = results.clone();

22   let worker1 = thread::spawn(move || {
       for i in 0 .. n/2 {
24         results.lock().unwrap()[i] = apply(jobs[i])
       }
26   });

28   let worker2 = thread::spawn(move || {
       for i in n/2 .. n {
30         results_copy.lock().unwrap()[i] = apply(jobs_copy[i]);
       }
32   });

34   worker1.join().unwrap();
   worker2.join().unwrap();
36 }

```

This program has a list of jobs to which a function `apply` should be applied. It does this in two parallel threads. The first thread applies this function to the first half of the jobs, the second thread applies this function to the second half of the jobs.

- (3 pts.) Explain why `jobs` and `results` are cloned.
- (4 pts.) The variable `results` has a far more complicated initialisation than `jobs`. Explain what this initialisation does, and why this is necessary.
- (3 pts.) Modify the program, such that the main thread after joining `worker1` and `worker2` can print out the final values of `result`.