2019-07-04 - Programming Paradigms - Concurrent Programming Test

Study: B-CS Programming Paradigms 201400537

In this exam, all questions are related to applications that would provide some support for the organisers or the coaches of the women worldchampionship football.

The exam will be open book. You can use the slides, course manual, the papers provided for Block 5 and 6 (all available with the test), and the book Java Concurrency in Practice, Goetz, Peierls, Block, Bowbeer, Holmes and Lea (2006). In addition, you are also allowed to use the Java API documentation: https://docs.oracle.com/javase/10/docs/api/overview-summary.html. The concurrency API java.util.concurrent is part of java.base.

Here you can find the slides for the Concurrent Programming lectures:

See attachment:	Lecture 1: Basics of Concurrency
See attachment:	Lecture 2: Synchronisation
See attachment:	Lecture 3: L veness, performance and fairness
See attachment:	Lecture 4: Homogeneous threading
See attachment:	Lecture 5: Safe concurrency
See attachment:	Lecture 6: F ne-grained concurrency, memory models

Here you can find the course manual: See attachment:

Here you can find the papers for Block 5:
See attachment: Using OpenMP - The Book

See attachment: Dr.Dobb's introduction to OpenCL

Here you can find the Rust documentation for Block 6:
https://doc.rust-lang.org/beta/book/

Number of questions: 4

You can score a total of 100 points for this exam, you need 55 points to pass the exam.

1 In this exercise, we will develop some code to manage a system that can display the playing scheme, as well as the results of the matches. The playing scheme denotes which game is played when, and once the game has started, it also depicts the current score.

To implement this application, we have a class Game See attachment: .

For each game, the names of the home and awayTeam are fixed, and should be readable at all times. During the match, the score might be updated. As multiple users could try to update the score simultaneously, this should be done in a thread safe way.

6 pt. **a.** Make Game thread-safe, using a blocking synchronisation mechanism. Avoid unnecessary synchronisation, and use 'guardedby' to indicate which synchroniser protects which field.

4 pt. **b.** Explain the performance considerations behind your solution.

6 pt. **c.** Make a lock-free, thread-safe version of your class Game.

4 pt. **d.** Sketch what the setHomeScore and setAwayScore methods would look like if Software Transactional Memory (STM) would be used, rather than using locks.

We also want to keep track of the order in which the goals are made. For some unknown reason, the decision is made to implement this in Rust.

3 pt. **e.** Below is the start of the Rust function to implement the system to keep track of the scores. (For simplicity, date is left out here.)

```
fn main() {
        let home_team = String::from("Netherlands");
        let away_team = String::from("Japan");
        let mut home_score = 0;
        let mut away_score = 0;
        let mut scores = Vec::new();
//...
}
```

Explain why home_score and away_score are mutable, while home_team and away_team are not.

8 pt. **f.** Finish the function, such that it has a thread that increases the home_score by 1, and adds 'h' to the scores vector, while the main thread increases the away_score by 1, and adds 'a' to the scores vector. It is allowed to change the types of the variables, if needed.

2 pt. **g.** Discuss what would change for an implementation using Software Transactional Memory (STM) if the score may be set only once.

9 pt. **h.** So far, we have assumed that the score may be continuously updated. An alternative is to set the score only once, at the end of the game. Provide a lock-free implementation of class SetOnce, which is a synchronizer that allows to set its field data exactly once. When a thread calls the update method, this returns true if the update succeed, otherwise (if the data is already set) it returns false.

```
public class SetOnce {
private int data;
public boolean update(int newVal) {
}
}
```

8 pt. **i.** The application that manages the playing scheme also stores the poule results, see class PouleResult See attachment: . For every poule, an instance of this class is created.In the 8th finals, the playing scheme is determined as follows:
Match 1: Winner group 1 - Second group 4
Match 2: Winner group 2 - Second group 3
Match 3: Winner group 3 - Second group 2
Match 4: Winner group 4 - Second group 1Now suppose we are writing a method determineNextMatch(int matchnumber),
which will ask for the different poule results, and fill in the new match. Suppose we execute determineNextMatch for match 1 to 4 in parallel. Discuss what is a potential concurrency risk for this code, illustrate it by an example, and explain how it can be avoided.

2 In this exercise, we will develop some code to do some statistics on the players. We assume every player participating in the world championships has a player_id, ranging from 0 to nrPlayers - 1. In addition, we assume we have an array scores, such that scores[i] indicates how many times the player with player_id i has scored. For simplicity, we will assume that the nrPlayers is a power of 2.

7 pt. **a.** Write an OpenCL kernel that can be used to find the player with the highest score. Your solution should work in log(n) time. (You may assume that your workgroup is sufficiently large to inspect the complete array).Hint: you could use some auxiliary array max_id, which stores the identifier of a local maximum. Make sure that it is clear where the result can be found eventually.

4 pt. **b.** Write a sequential program that checks for every player where she scored just as many times as the player with the next player id. If this is the case, write a 1 in a result array, otherwise write a 0. Use OpenMP to indicate how this program could be parallellised.

4 pt. **c.** Write a sequential program that counts the total number of players that scored at least one goal. Use OpenMP to indicate how this program could be parallellised.

3 One of the team players is studying Computer Science (in addition to playing football). Help her to make the following exercise.

Consider the following implementation of a NonReentrantSpinLock.java

See attachment:

4 pt. **a.** What would happen with this implementation if a thread calls unlock() while not holding the lock.

3 pt. **b.** Discuss why the unlock() does not use a compareAndSet operation.

8 pt. **c.** Adapt the NonReentrantSpinLock in order to make it reentrant. It should work correctly under the same conditions as the non-reentrant-spinlock.

4 In this exercise, we will develop an application that can be used by the coaches to decide the line-up for the team (i.e. who will play on what position). A first attempt to develop such a class is made, see the class TeamStrategy (with associated enum Position).

See attachment: See attachment:

2 pt. **a.** What is wrong with this class if it would be used in a concurrent setting, i.e. if multiple people from the support staff would call the method playerAt simultaneously.

12 pt. **b.** Provide 3 different solutions to make this class thread safe. Make sure that at least one of your solutions is lock-free.

6 pt. **c.** For each of your solutions, discuss one advantage and one disadvantage.