Exercises week 13

Last update 2020-11-18

Goals

The goals of this week is that you:

- get a basic understanding of *Lock free coordination*,
- try Git tools for merge/rebase,
- understand the implications of the CAP theorem,
- understand how *operation transform* works on text files and
- get to know about *operational transform* in distributed databases.

Do this first

Read the papers and see the videos listed in the Readme file for week 13.

Exercise 13.1 In the directory `Code/GitEx` you find the two text files used to explain Git merge/rebase in the lecture.

1. *Green* Make a local copy on your PC of the directory `Code/GitEx`, and make this local directory a Git repository.
2. Redo the changes to your local Git repository that was done in the lecture (slide 4 ff).
3. Write a small explanation of what happens in the last step (git merge ...).
4. *Yellow* Create a new branch (`numberletters`) in you Git repository. In this branch change the, contents of the file `numbers.txt` to be `12xy54p`. Switch back to the master branch. What would happen if you tried the command `git rebase newnumbers`? What would you need to do in order to make this rebase work?

Exercise 13.2 *Green* This website illustrates how operational transform works on simple texts: `https://operational-transformation.github.io`.

Go through these steps and write down an explanation of what happens.

1. Delete the string "Lorem ipsum" in both Bob's and Alice's windows.
2. Push the small arrows until both windows show the same string.
3. Enter the string "Introdction" in Alice's window.
4. Push the small arrows until both windows show the same string.
5. Correct the spelling mistake (missing "u") in Bob's window.
6. Push the small arrows until both windows show the same string.
7. Enter the string "This is a short explanation" into Alice's window.
8. Enter the string "This explains " into Bob's window.
9. Push the small arrows until both windows show the same string.

Yellow

10. Start again from step 6 (in the green part of the exercise).
11. Enter the string "This ex" in Bob's window.
12. Enter the string "This is" in Alice's window.

Does the final result depend on the order in which you push the arrows? Explain why or why not?

Exercise 13.3 In the lecture, the CAP theorem was discussed. Operational transform is an example of how to handle synchronization in a system where either A(vailability) or P(artition tolerance) can not be guaranteed. *Green*

1. Give an example of a distributed system where A(vailability) cannot always be guaranteed.
2. Give an example of a distributed system where P(artition tolerance) cannot always be guaranteed.
3. Give an example where *Strong eventual consistency is different than Strict conistency.*

Exercise 13.4 In this exercise, you will develop a new implementation of the BoundedBuffer from week 3. The new buffer is described on page 475 and 476 of the paper "Linearizability: A Correctness Condition for Concurrent Objects" by Herlihy and Wing (included in the readings for week 13). We will call this buffer a WingBuffer (after the second author of the paper).

The key idea of the WingBuffer is that it has very minimal locking and yet is *linearizable* (introduced in lecture 13) and formally defined in the paper "Linearizability: ..." referred to above.

1. *Green* Implement a version of the `Wingbuffer` that uses a datastructure (array-like) with two atomic (`synchronized`) operations: `INC` and `SWAP`. Here is a skeleton:

```
class WingStructure<T>{
  // TO DO datastructure containing FIFO
  public synchronized <T> INC(){
    //TO DO implement
  }
  public synchronized <T> SWAP(int i){
    //TO DO implement
  }
}
```

   Make the `//TO DO`s and ensure that the resulting class can be compiled (with any syntax errors). If you find it simpler, you may omit the type parameter <T> and change the type elements of the FIFO to `int`.

2. *Yellow* Use the `WingStructure` you just made to write a Java implementation of the FIFO based on this skeleton:

```
class WingBuffer {
  static int iNull= //To DO find a representation of null

  private final WingStructure<T> rep;

  public int Deq {
    //TO DO implement
  }
  public int Enq {
    //TO DO implement
  }
}
```

   The `WingBuffer` must **not** use any additional locking. Write a small program testing that the sequential functionality of the FIFO is correct. You need not check the locking.

3. Make a new implementation of the `WingBuffer` (from the first question) without `synchronized`. Instead you should use the atomic operations provided in:
   `java.util.concurrent.atomic.AtomicIntegerArray` and
   `java.util.concurrent.atomic.AtomicInteger` to implement `INC` and `SWAP`.