UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS226 - Concurrent systems

Practical 3 Specifications - Registers

Release date: 19-08-2024 at 06:00
Due date: 23-08-2024 at 23:59
(FF will remain open until 25-08-2024 23:59 for technical difficulties)
Total marks: 12

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually; no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as no extension will be granted.
- If your code does not compile, you will be awarded a mark of zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- If your code experiences a runtime error, you will be awarded a zero mark. Runtime errors are considered unsafe programming.
- Read the entire specification before you start coding.
- **Ensure your code compiles with Java 8**
- The usage of ChatGPT and other AI-Related software is strictly forbidden and will be considered as plagiarism.

2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent), and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then choose the Plagiarism option under the Services menu). **If you have any form of question regarding this, please ask one of the lecturers to avoid any misunderstanding.** Also note that the OOP principle of code reuse does not mean that you should copy and adapt code to suit your solution.

3 Outcomes

On completion of this practical, you will have gained experience with the following:

- Registers
  - Identifying valid reads.

4 Introduction

4.1 Registers

In concurrent computing, registers serve as a fundamental building block for inter-process communication and synchronization. A register is a small amount of storage that can be read and written to by concurrent processes or threads. The primary challenge with registers in a concurrent environment is ensuring that operations on them are atomic, meaning that each operation is completed entirely or not at all, preventing race conditions. Registers are often used to store simple variables like flags, counters, or shared data among threads, and they require careful design to ensure that their state remains consistent and correct despite concurrent access. Advanced techniques, such as compare-and-swap (CAS) and other atomic operations, are often employed to manage these challenges effectively in concurrent programming.

5 Tasks

You are tasked with creating a method to find all the legal reads of a certain type of SRSW concurrent register.

A `RegisterOperation` has two variants: READ and WRITE. Register operations also have a start and end time. This will be used to determine which Register Operations overlap and therefore which read values are valid.

The `.toString()` method is given for your convenience.

5.1 Task 1 - getValidReadValues

Implement the `getValidReadValues` method in the `Register` class which finds all the legal reads after a sequence of `RegisterOperations`. This method uses the List of type `RegisterOperation` and returns a List of Lists of type `Integer`. Each sub-list should be a list of legal read results. The List of Lists should contain every combination of legal read results. With this specific type of register reads that overlap writes are able to flicker between values non-atomically. See figure 4.3 in the text book. This is very important.

For example: given the following `RegisterOperations`:

```
Register register = new Register();
     register.write(0, 1, 2); // W(0)
     register.read(2, 4); // R1
     register.read(4, 6); // R2
     register.write(1, 5, 8); // W(1)
     register.read(7, 9); // R2

    List<List<Integer>> validReads =
         register.getValidReadValues();
```

The `validReads` super list should contain four sub lists of `Integer`s all of which contain valid read values for the associated read . The lists are:

```
[[0,0,0],[0,0,1],[0,1,0],[0,1,1]]
//R1 can read only 0
//R2 can read 0 or 1
//R3 can read 0 or 1
```

The order of the sub-lists within the super list is not important. The marker would also mark the following as correct:

```
[[0,0,1],[0,1,1],[0,1,0],[0,0,0]]
```

NB: While reads and writes may overlap, reads may never overlap with other reads and writes may never overlap with other writes. Moreover, only assume overlapping when the overlap is obvious. The ambiguous case when one operation stops at time=4 and another starts at time=4 should not be considered an overlap.

Note: A very generous main has been provided. I know you're writing COS214 this week. Good Luck!

6 Upload checklist

The following files should be in the root of your archive

- `Register.java`
- `Main.java` will be overridden
- `RegisterOperation.java` will be overridden

7 Allowed libraries

These libraries will be allowed by the automaker but you must still do the given tasks with your own code (i.e you may not use the libraries to complete the task for you)

- `import java.util.*` needed for List and Arrays but can be used for more ;)

8 Submission

You need to submit your source files on the FitchFork website (https://ff.cs.up.ac.za/). All methods need to be implemented (or at least stubbed) before submission. Place the above-mentioned files in a zip named uXXXXXXXX.zip where XXXXXXXX is your student number. Your code must be able to be compiled with the Java 8 standard.

For this practical, you will have 20 upload opportunities and your best mark will be your final mark. Upload your archive to the appropriate slot on the FitchFork website well before the deadline. **No late submissions will be accepted!**