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 2 Specifications - Sequential Consistency

Release date: 12-08-2024 at 06:00

Due date: 16-08-2024 at 23:59

(FF will remain open until 18-08-2024 23:59 for technical difficulties)

Total marks: 22

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:

- Sequential Consistency
  - Identifying execution orders which are sequentially consistent.

4 Introduction

4.1 Sequential Consistency

Sequential consistency is a fundamental concept in concurrent programming that ensures the correctness and predictability of operations performed by multiple threads. In a system that guarantees sequential consistency, the results of the execution of operations by various threads appear as if the operations were executed in some sequential order. Moreover, the operations of each individual thread are ordered according to the thead-order.

This means that even though the threads might execute concurrently in reality, the system must ensure that the final state of the system reflects a logical sequence where all operations from all threads are consistent with a single, global order. Sequential consistency simplifies reasoning about concurrent programs because it ensures that the interleaving of operations does not result in unexpected or erroneous behaviour.

5 Tasks

You are tasked with creating a method to find all the sequentially consistent orders of a list of `MethodCalls`.

A `MethodCall` has a `threadId`, an `orderInThread` and an `action`.

- **threadId**: This indicates which thread calls the method. For example, a value of "A" indicates that thread A called/executed the method.
- **orderInThread**: This indicates the order in which the thread with **threadId** called the method. For example, a value of *1* indicates that the thread called this method first. A value of *2* indicates that this method was called second etc.
- **action**: This is the action the the method performs. For this practical it can be one of: "enq(x)" or "deq(x)" where "x" can be any single letter. For example "enq(a)", "deq(z)" and "enq(q)" are all valid actions but "enq(it)" and "deque(x)" are not.

The `MethodCall` class has a `.equals()` method and a `.toString()` method. These methods are used for marking and should not be changed.

5.1 Task 1 - Execution Order Checker

Implement the `findPossibleOrders` method in the `ExecutionOrderChecker` class which finds all the sequentially consistent execution orders of a given List of `MethodCall`s. This method receives a List of type `MethodCall` as an input (the `MethodCall` class has been provided) and returns a List of Lists of type `MethodCall`. Each sub-list should be a sequentially consistent execution order of the method calls. The List of Lists should contain all (and only) the sequentially consistent execution orders of the given `MethodCall`s.

For example: given the following List of `MethodCalls`:

```
List<MethodCall> operations = Arrays.asList(
            new MethodCall("A", 1, "enq(x)"),
            new MethodCall("A", 2, "deq(y)"),
            new MethodCall("B", 1, "enq(y)"),
            new MethodCall("B", 2, "deq(x)")
    );

    List<List<MethodCall>> possibleOrders =
     ExecutionOrderChecker.findPossibleOrders(operations);
```

The `possibleOrders` super list should contain two sub lists of `MethodCall`s both of which are sequentially consistent execution orders (these are also the only sequentially consistent execution orders). The lists are:

```
[(B,1,enq(y)), (A,1,enq(x)), (A,2,deq(y)), (B,2,deq(x))]
[(A,1,enq(x)), (B,1,enq(y)), (B,2,deq(x)), (A,2,deq(y))]
```

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

```
[(A,1,enq(x)), (B,1,enq(y)), (B,2,deq(x)), (A,2,deq(y))]
[(B,1,enq(y)), (A,1,enq(x)), (A,2,deq(y)), (B,2,deq(x))]
```

NB: method calls must appear in the correct "thread-order" and method calls must follow a FIFO queue structure within each list. In other words, if ("A",1,"enq(x)") is called before ("B",1,"enq(y)") then ("B",2,"deq(x)") must be called before ("A",2,"deq(y)").

6 Upload checklist

The following files should be in the root of your archive

- `ExecutionOrderChecker.java`
- `Main.java` will be overridden
- `MethodCall.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!**