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 1 Specifications - Basic Locks

Release date: 29-07-2024 at 06:00
Due date: 02-08-2024 at 23:59
(FF will remain open until 04-08-2024 23:59 for technical difficulties)
Total marks: 100

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:

- Basic Locks
  - Peterson Lock
  - Filter Lock
  - The Java Lock interface

4 Introduction

4.1 Locks

In concurrent programming, locks are essential synchronization tools that control access to shared resources by multiple threads, ensuring data integrity and preventing race conditions. By allowing only one thread to access a resource at a time, locks such as mutexes and read/write locks help maintain consistency and atomicity in multi-threaded applications. In Java, custom locks can be made be using `implements lock` from the java.util.concurrent.locks package.

4.2 Scenario

A task which often requires various levels of synchronisation is marking test papers. In this scenario, there will be three markers and one shared pile (Queue) of papers. In order to mark a paper a marker should take a paper from the pile and mark it (by calling the `paper.mark()` method). Sometimes two (or more) markers will reach for the top paper at the same time. This contention requires some sort of synchronisation.

5 Tasks

You are tasked with extending the standard Peterson Lock to handle three threads and then using the extended lock to control access to a shared queue of papers.

5.1 Task 1 - Threeterson

A skeleton for the `Threeterson` lock has been provided. The `Threeterson` class implements the `Lock` interface and you are responsible for implementing the `lock` and `unlock` methods. Your approach should use victims and levels much like the Filter Lock.

Your lock should add text to the given output object:

- When a thread enters a level greater than 0 you should add: Thread.currentThread().getName() + " at Ln" (where n is the level)
- When a tread becomes the victim of a level you should add: Thread.currentThread().getName() + " is the victim of Ln" (where n is the level)
- When a thread acquires the lock you should add: Thread.currentThread().getName()+" has the lock". (This can be done within the thread (i.e the Marker class) or in the lock.
- When a thread releases the lock you should add: Thread.currentThread().getName()+" unlocked the lock". (This can be done within the thread (i.e the Marker class) or in the lock.

You may use the given ReentrantLock (named printLock) to synchronise adding to the output and changing the lock state.

There are some restrictions that your implementation should abide by:

- Your lock must work for any thread names which are unique mod 3. i.e You could get threads 0, 1, 2 or threads 9, 10, 11 or threads 99, 100, 101 etc
- While you may use them for inspiration, you may not copy-paste the Peterson or the Filter Lock from the textbook.
- A thread may not claim the lock while it is held by another thread.
- A thread may not release a lock it does not hold.
- A thread must pass through all the levels before claiming the lock.
- A thread must at some point be the victim of every level it passes through.
- A thread may not pass to the next level if another/other thread/threads are at that level and it is the victim of that level.

5.2 Task 2 - Marker

A skeleton for the `Marker` class has been provided. Each Marker should be initialised with a shared Threeterson lock, a shared queue of Papers and a shared Output (The `Paper` and `Output` classes have been provided). In the `run` method, a Marker should pop a Paper off the shared queue and then mark it (`paper.mark()`). This should continue while there are still papers to mark. You will need to use the Threeterson lock to control access to the shared paper queue and shared output object.

6 Marking

The marking for this practical will work as follows:

- You will implement the tasks and upload your solution to Fitchfork.
- The OutputValidator class will be overridden by FitchFork and will then be used to mark the output object that is produced by your code à la

  ```
  Output o=new Output();
  Threeterson lock=new Threeterson(o);
  Queue<Paper> allPapers = new LinkedList<>();
  Marker m = new Marker(lock,o,allPapers); //example
  // do setup, threading and running
  OutputValidator validator = new OutputValidator(o);
  boolean isValid = validator.validate();
  ```

- You will receive a mark on Fitchfork
- A mark on Fitchfork of greater than 0 will allow you to go to a practical session to be marked by a tutor.
- In the practical session a tutor will ensure that your implementation has kept within the restrictions (i.e no copy-pasting, not using libraries to do the whole task). The tutor will then assess your understanding of your implementation and the practical content.
- You will receive a final mark which will be some weighted combination of your Fitchfork mark and your "understanding" mark.

7 Upload checklist

The following files should be in the root of your archive

- `Main.java` will be overridden
- `Marker.java`
- `Output.java` will be overridden
- `OutputValidator.java` will be overridden
- `Paper.java` will be overridden
- `Threeterson.java`

8 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)

- `java.util.Queue` for Marker
- `java.util.concurrent.ConcurrentLinkedQueue` for Output
- `java.util.Random` for Paper

- `java.util.concurrent.TimeUnit` for Threeterson
- `java.util.concurrent.locks.Condition` for Threeterson
- `java.util.concurrent.locks.Lock` for Threeterson
- `java.util.concurrent.locks.ReentrantLock` for Threeterson

9 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!**