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 6 Specifications - Synchronous Sets

Release date: 07-10-2024 at 06:00
Due date: 11-10-2024 at 23:59
(Submission will remain open until 13-10-2024 23:59 for technical difficulties)
Total marks: 36

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:

- Synchronisation: Course-Grained, Fine-Grained and Optimistic

4 Introduction

4.1 Synchronisation

In concurrent programming, synchronization techniques like fine-grained, coarse-grained, and optimistic synchronization are crucial for managing access to shared data structures, such as sets, among multiple threads. Fine-grained synchronization involves using multiple locks, each protecting a small portion of the set. For example, in a linked list-based set, each node might have its own lock. This method allows for high concurrency, as multiple threads can perform add(), remove(), and contains() operations on different parts of the set simultaneously. However, it adds complexity and the risk of deadlock due to the careful lock management required.

Coarse-grained synchronization, on the other hand, uses a single lock for the entire set. This approach simplifies implementation since only one lock is needed, but it limits concurrency. When a thread acquires the lock, other threads must wait, potentially resulting in performance bottlenecks, especially during frequent add(), remove(), or contains() operations.

Optimistic synchronization provides a different approach by allowing threads to proceed with operations without initially acquiring a lock. Instead, it performs checks at the end of the operation to detect conflicts. For instance, in add() or remove(), a thread checks if the set's state has changed before committing its changes. If a conflict is detected, the operation is retried. This method can improve performance in scenarios with low contention, but it may result in increased retries if conflicts occur frequently.

Each synchronization strategy has its strengths and weaknesses, and their effectiveness depends on factors like contention levels and the specific use case.

5 Tasks

You are tasked with implementing three "Set" classes using coarse-grained, fine-grained and optimistic synchronisation. After implementing them, you must compare the implementations in various scenarios (at least 3). The goal is to determine the use cases of each type of synchronisation.

5.1 Task 1 - Lock implementation

Implement the "Set" classes using `coarse-grained`, `fine-grained` and `optimistic` synchronisation.

5.2 Task 2 - Scenarios

Create at least 3 scenarios to test your synchronisation implementations. These scenarios should show the strengths and weaknesses of each type of synchronisation. Record any relevant metrics over any relevant parameters (e.g. time taken vs number of threads; average time spent waiting vs number of method calls; etc.) Hint: A sample size of 1 is not recommended.

5.3 Task 3 - Report

Write a (very) short report which includes:

- Your name and student number
- An introduction which reads like a storybook (have fun)
- Graphs showing your findings of time vs number of threads (and any others you can think of)
- Explanations of the trends in your findings
- Reasons for the trends. (E.g. fine-grained is better in this scenario because it has more locks)
- Conclusions as to when each type of synchronisation is best used.
- References to any resources you used
- The report recipe:
  - A list of materials needed to produce this report (these need not be boring e.g. a cup of knowledge, a dash of fun).
  - The steps required to produce this report (using the above materials)

6 Mark Breakdown

- Course-Grained Synchronisation - 5 marks
- Fine-Grained Synchronisation - 5 marks
- Optimistic Synchronisation - 5 marks
- Scenarios - 6 marks
- Report - 15 marks
- **Total - 36 marks**

7 Upload checklist

Upload all the files, including your report, that you need for your demo in a single archive.
NB: Submit to the ClickUp Module, there will be no FF submission.

8 Allowed libraries

These libraries will be allowed 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.*`

9 Submission

You need to submit your source files on the ClickUp module website. Place all the necessary 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.

Upload your archive to the appropriate practical on the ClickUp website well before the deadline. **No late submissions will be accepted!**