Programming Assignment for Concurrency & Multithreading

Concurrent Data Structures

Introduction

In this assignment you will create several concurrent data structures in Java. The learning goal of this assignment is:

- Correctly implement concurrent data structures.
- Assess the performance of these data structures.

Details of the assignment

You are required to write several concurrent data structures in Java. Both Bachelor students and Master students need to write a sorted CoarseGrainedList, CoarseGrainedTree, FineGrainedList, and FineGrainedTree. Master students have to implement an additional LockFreeList and LockFreeTree. **Watch out!** This assignment can be very challenging.

The book by Herlihy and Shavit [2] provides example implementations on several of these data structures. However, it is not clear which data structures are right for which situations. Therefore, you need to assess the performance of each data structure.

We have provided a driver program that will first enter N elements in random order and will then remove the same N elements in a different random order. After this, the data structure should be empty.

As a first step, you should hand in a design-document in which you give details on your approach to implement the data structures. Focus mainly on the FineGrainedTree (Bachelors and Masters) and the LockFreeTree (Masters). In this design-document we also ask you to form hypotheses about the performance and scalability of each data structure. Try to estimate the performance of the data structures by taking into account the number of elements in the data structure, the number of threads operating on the data structures, and the amount of work a thread is doing in comparison to inserting/removing an element. Make sure you support your claims well. After about one week you will receive feedback about your document.

Next, you can start implementing the data structures. All data structures implement the `Sorted` interface which provides the methods `add(T t)` and `remove(T t)`. The data structures need to be sorted according to the ordering of `Comparable<T>`. We provide a basic skeleton of the program that uses the discussed data structures. It is your task to implement the `add(T t)`, `remove(T t)`, and `toString()` methods.

There are two main differences between the data structures in the book and the data structures we ask you to implement. First, the list data structures presented in the book are sorted according to the hash value of the stored items, whereas we ask you to sort the list according to the `Comparable<T>` interface. Furthermore, the data structures in the book do not allow double elements, whereas the linked lists and binary search trees in this assignment should.

The following paragraphs discuss the data structures in more detail.

CoarseGrainedList This CoarseGrainedList needs to be a singly-linked list. The elements are ordered in a monotonically non-descending order. This means that double elements can occur in the list. Adding or removing an element locks the complete data structure. Figures 9.4 and 9.5 in [2] show examples of update methods of a linked list with coarse-grained locks.

CoarseGrainedTree This tree is a binary search tree that also allows double elements. Elements are stored at internal nodes and leaf nodes of the tree. The complete data structure needs to be locked when an element is added or removed.

FineGrainedList This linked list should not lock the complete data structure, but only parts of it to allow concurrent updates on different parts of the list. Figure 9.6 and 9.7 in [2] show examples of updates on a list with fine-grained locks.

FineGrainedTree This binary search tree is similar to a CoarseGrainedTree, but uses fine-grained locks, so locks at the nodes of the tree. Multiple concurrent updates on different parts of the tree should be allowed.

LockFreeList (Master students) The following two data structures are only meant for Master students. This lock-free linked list should not use locks, but compare-and-set instructions to give freedom of blocking. Section 9.8 in [2] discusses an example of a lock-free list.

LockFreeTree (Master students) We ask Master students to also implement a lock-free binary search tree. This is a challenging task, and we therefore drop the requirement of allowing double elements in the tree. Ellen et al. discuss a leaf-based (values are only stored in leaf nodes) implementation in [1]. We ask you to implement this lock-free binary search tree. Follow the algorithm as described in the paper.

Finally, you will evaluate the performance of your data structures using a multi-core machine with at least 8 cores. You can make use of several compute servers:

- `flits.few.vu.nl`
- `fluit.few.vu.nl`
- `galjas.few.vu.nl`

In your evaluation document you assess the performance of the data structures and compare them to your hypotheses in the design-document. Make sure you explain the results you obtain and support them with numbers, for example with graphs. Show how your data structures perform under different levels of concurrency.

Take note! There will be one question in the final exam about the programming assignment. You will receive feedback about the assignment on October 19th at the latest, so you can incorporate the feedback in your study of the material for the exam.

Timeline

- **Design-document**
  - details of implementation approach
  - hypotheses on performance

  Deadline on Wednesday September 12th at 23.59
  Feedback received by Wednesday September 19th

- **Implementation and evaluation**
  For all four (Bachelors) or six (Masters) data structures:
  - implementations of `add(T)`, `remove(T)` and `toString(T)`
  - evaluation of performance

  Deadline on Wednesday October 10th at 23.59
  Feedback received by Friday October 19th

Assessment

First, your grade depends on the design document and how well your arguments are for your hypotheses. Second, your grade depends on the quality of your implementation and thoroughness of your evaluation.

Please make sure that your code is easily understood and uses a clear coding style. It is important that you explain the results when evaluating the performance of your data structures.

Do not hesitate to send an email if you have any questions or want to make an appointment. See the submission section for email addresses. You can ask questions in either English or Dutch.

Submission

The design document and the evaluation document should be in PDF format. Your java source files need to be compressed in a single archive. Send the PDFs and archive to `h.p.hijma@vu.nl` and `s.j.j.vijzelaar@vu.nl`.

You are allowed to do the assignments alone or in pairs. Groups become fixed after submitting the design document. Always list both the VU-net-ID and student-number of you and your partner (if applicable) in your submissions.

Deadlines are strict! If you are unable to finish the assignment in time, at least turn in your unfinished work before the deadline.

References

[1] Faith Ellen, Panagiota Fatourou, Eric Ruppert, and Franck van Breugel. Non-blocking Binary Search Trees. In *Proceeding of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, pages 131–140, New York, NY, USA, 2010. ACM.

[2] M. Herlihy and N. Shavit. *The Art of Multiprocessor Programming.* Morgan Kaufmann Pub, 2008.