COMP3151/9154 T2 2023
Assignment 1
Total marks: 20
Due Date: 5pm Fri Jul 14, 2023

By default, this assignment should be done by a team of *two* of students, but no more. Submissions by individuals will not be penalised, but of course, this means you have to do more work. If you are lacking a partner, advertise on the forum and/or in tutorials to find yourself a partner.

1 The problem

We would like to implement a data structure that maintains a large set of non-negative numbers, that can be accessed concurrently by multiple processes, which perform the following types of operations:

- `insert(x : int)` — insert a positive number `x` into the set.
- `delete(x : int)` — remove a positive number `x` from the set, if it is present.
- `member(x : int) : bool` — determine whether the positive number `x` is an element of the set.
- `print_sorted()` - print out the set of elements in sorted order.

Note that a number is either a member of the set or it is not a member - there is no notion of duplicated copies of the same number (this would be a bag rather than a set.) It is not known in advance how many processes will be trying to use these operations concurrently, so we require a solution that works for an arbitrary number of processes.

The amount of memory for the data structure is bounded, so that it can contain at most N elements. Any process attempting to insert a new number when there are already N members should be blocked until deletions reduce the number of elements to be less than N.

To the largest degree possible, we would like to allow these operations to happen concurrently. In particular, at a minimum, we want to allow multiple processes to be performing the `member` operation at the same time, since this requires only reading the data structure. However, where opportunities exist to run any of the types of operations concurrently (running on multiple cores in a multicore system), we would like to take advantage of them.

For performance reasons, we want the calls to `member`, `delete` and `print_sorted` to be as efficient as possible. Assume that the `insert` calls are expected to be less frequent, and that it is acceptable for these operations to be slower (run in time $O(N)$).

Based on these requirements, it has been decided to implement the data structure using an array that is maintained in sorted order, and to implement the `member` operation using a binary search algorithm.[1] Since N could be very large, the array could be so large as to consume most of memory, and it should *not* be assumed that there is sufficient space to allow copying of the array. All operations should therefore be performed "in place", using only the space allocated to the array and a small constant number of local variables and additional shared variables for managing concurrency.

It has been noted that there is an opportunity to implement the `delete` operation in a way that makes it very efficient. Since all numbers to be stored are positive, rather than reorganise the array when deleting an element, it suffices to mark the location as deleted by writing a negative number to the location storing it. This does require some modification to the binary search algorithm, since the search may now land on deleted locations, requiring that adjacent locations be inspected to look for an undeleted element to determine how to proceed with the search. Moreover, if there are many deleted entries, binary search will operate less efficiently, and space is being "wasted". Therefore, an *insert* operation should contribute to cleaning up the deleted entries and "compactify" the region of the array that it modifies, so that it forms a contiguous block of actual members of the set. (Note that *insert* will need to shift some entries in the array in order to maintain it in sorted order.) For purposes of cleaning up the array when there are deletions but no *insert* operations for a long period, it is also desirable to have a `cleanup`() operation that can be called periodically to do this work.

It is desired that the implementation be as portable as possible between different processor architectures, so the solution will be programmed in Java, and use of any processor-specific hardware instructions has been excluded. (You may, however, assume that the hardware works in such a way that a read concurrent with a write returns either the value written or the value just before the write — simultaneous writes and reads do not cause random values to be returned.)

Note that, in general, we would like to maximize the amount of concurrency, and ensure that the answers returned reflect the insertions and deletions into the set as quickly as possible, but we may be prepared to sacrifice some recency of results in the sense that insertions and deletion operations that have not yet run to completion are ignored in the results of `member` and `print_sorted`.

[1]An alternative to an array would be to do the implementation using tree-based data structures that support efficient search. This is possible, but more complex, so stick to an implementation based on an array.

2 Deliverables

1. **A report** that describes your solution. In particular, this should contain:

   - The names and student numbers of your team members.
   - What conflicts between operations does your solution have to address, which could result in damage to the data structure or incorrect outputs?
   - Give a precise specification of the safety and liveness properties your solution guarantees.
   - How you have used Java concurrency control to prevent to implement the specification. Include a precise statement of the semantics of the constructs you use.
   - What is the actual big-O time complexity of each of the operations of your implementation? Have you implemented any optimization ideas beyond the ones already suggested above, in order to maximize the amount of concurrency in your solution?

   The above points are worth **(4 marks)**. Further sections of the report, for which additional marks are allocated separately, are stated below.

2. **(4 marks) Promela models and specifications** that you have developed to convince yourself that your design for synchronization of the concurrent operations works correctly, and satisfies your specification.

   Note that it may not be necessary for the Promela models to contain the full details of your code, particularly as it relates to maintaining the array. (Consider the way that we abstracted the idea of "critical section" in lectures - for properties like mutual exclusion it does not actually matter what this code does.) Your report should explain how your Promela code relates to your implementation, however.

3. **(4 marks) Proofs:** Marks will be given for inclusion of proofs of correctness of key ideas in your solution. As much as possible, *justify* the correctness of your solution. (If you do not find enough time for a full proof, try at least to identify important invariants and assertions holding at code locations, and give informal explanations of why they are preserved. Marks will be given for partial results, e.g., showing that the binary search correctly interacts with concurrent deletions.)

4. **(5 marks) A Java implementation** of your design. You are free to use any Java concurrency constructs, but note that this assignment is asking for a finer level of concurrency control and better performance than simple mutual exclusion of all the required functions.

5. **(3 marks) Test Scripts:** You should do careful testing of your implementation. Explain in your report the approach to testing that you have

applied, and summarise your test scripts. In particular, explain in your report how your testing deals with the large number of potential interleavings of concurrent operations. In what sense can it be considered to be "complete"? Can you justify, by experimental data, that your solution provides a speedup over a simpler implementation based on the reader-writer solution discussed in class?

Recommendation: It is not advised that you split the work on this project by having one person develop the Promela models and/or proofs and the other responsible for coding the Java. This is *highly likely* to result in inconsistencies between the two! Begin by working together on your design for managing concurrency, and develop the Promela models and proofs together until you have these correct. Then work together to implement and test your solution in Java.

3 Plagiarism

This assignment is subject to the usual UNSW rules on plagiarism. (See `https://www.student.unsw.edu.au/plagiarism/integrity`.) All components of the submission should be the original work of your team. You may use Java libraries that are part of the Java language distribution, but not code from other sources. Where you use ideas sourced from others, give full credit and references in proper academic style.

Use of generative AI in this assignment is prohibited. You may use spelling and grammar correction tools, but not auto-translation tools (it should be noted these have a risk of generating text that appears to have been AI generated!) Use of auto-translation will not be considered an acceptable excuse in case we conduct an investigation for use of AI.