Concurrent Algorithms and Data Structures
Lab Assignment 1

Parosh Aziz Abdulla
Fridtjof Stoldt
Tage Johansson

November 11, 2024

Deadline: 2023-12-04

1 Instruction For The Lab

1.1 Lab Template

There is a code template for this lab. You are not required to use the template, but it is highly recommended, as the tasks are defined using the template. The `README.md` file contains a quick overview of the contained files.

1.2 Compilation

The lab template contains a `makefile` file that can be used to build the project (Using `make`). The file has been tested on Ubuntu with g++ v11.4.0, but it should also work on macOS and other Unix-based systems. You can also manually invoke the following `g++` command in the `src` folder.

```
g++ -Wall -lpthread -std=c++20 main.cpp
```

This will generate an `a.out` file.

Your final submission should compile and run on the Linux lab machines from Uppsala University[1]. If your submission requires a different command for compilation, please document it in your report.

1.3 Run Instructions

The created binary takes the task number as the first argument. For example `./a.out 1` will run the first task.

1.4 Memory Management

C++ uses manual memory management. In concurrent programs, this can cause problems if one thread uses a pointer to access memory freed by another thread. If it is possible, try to free all the memory you allocate. Otherwise, you can skip the freeing if you explain in your report when freeing would be unsafe.

[1] See `https://www.it.uu.se/datordrift/maskinpark/linux` (Accessed 2024-10-25) for more information

2 The `Set` Abstract Data Type

Abstract Data Types (ADTs) are mathematical objects that allow us to specify the expected behaviors of implementations of common data structures such as sets, queues, and stacks. The `Set` data type used in this lab has the following methods:

1. `add(elem) -> bool`:

   If `elem` is not in the set, it will be added, and `true` is returned; otherwise `false`.

2. `rmv(elem) -> bool`:

   If `elem` is in the set, it will be removed, and `true` is returned; otherwise `false`.

3. `ctn(elem) -> bool`:

   If `elem` is in the set `true` is returned, otherwise `false`.

Task 1 Standard Non-Concurrent Set (10 Points) An operation is defined as a pair of a method and an argument to the method. An operation together with the output can be written as a tuple, such as `(add, 6, false)` when the operation `(add, 6)` returns `false`. In the given lab template, these are called `Events`. From a given state s, an event is allowed if the operation yields the expected output. For instance, assume we have the set $s = \{3, 7, 9\}$. The program allows the events $(\texttt{add}, 5, \texttt{true})$ and $(\texttt{add}, 7, \texttt{false})$ from s, but not $(\texttt{add}, 8, \texttt{false})$.

The lab template implements the set abstract datatype, where the state is an instance of C++ `std::set`. In this task, you are supposed to test the implementation of the set abstract datatype by running event sequences.

Run task 1 of the lab template using the command `./a.out 1` and inspect the output. Make sure that you understand the code of `task_1(...)`.

Your task is to add the following sequences:

1. Add a valid sequence of 10+ instruction
2. Add a sequence that fails an `add(3)` after 5 instructions
3. Add a sequence that fails a `contains(3)` after 5 instructions
4. Add a sequence that fails a `remove(3)` after 5 instructions

The places you need to modify are marked with \\ `A01:` comments

Task 2 A Simple Set (20 Points) Implement a simple `Set` abstract datatype in the `simple_set.hpp` file, where the state is a linked list. The places you may need to modify are marked with \\ `A02:` comments. You can also add testing and debugging code. Please do not use any synchronization mechanism for this set.

You need to test your implementation. For this, insert all performed operations with their output into an event sequence shared across all threads. The template provides the `EventMonitor` to accept events and test them. A separate thread is used to monitor the sequence and check that each event is allowed. If an operation is not allowed, a message will be printed. Make sure to keep the `this->monitor.add(...)` code for monitoring.

You can run the code for task 2 using the command `./a.out 2`. The single-threaded version should process all operations just fine. The multi-threaded version should fail, which is expected.

In your report, give at least one example of how concurrent access may lose data or crash the system with this simple data structure.

Task 3 A Coarse-Grained Set (30 Points) Implement a `Set` using a linked list that uses coarse-grained locking internally to allow concurrent access. This implementation should be done in `coarse_set.hpp`. The places you may need to modify are marked with `\\ A03:` comments.

For this data structure, you also want to test our implementation. For this, you need to identify the linearization points and insert the corresponding event into the `EventMontior`. Document the linearization policy in your report. The rest of the testing code is already present in the `task_3()` function of the template. You can run the code of `task_3()` using the command `./a.out 3`.

Hint: You might be able to reuse parts from task 2.

Task 4 A Fine-Grained Set (40 Points) Implement a `Set` using a linked list, that uses fine-grained locking internally to allow concurrent access. This implementation should be done in `fine_set.hpp`. The places you may need to modify are marked with `\\ A04:` comments.

Test your implementation like the previous task. Identify the linearization points and insert the corresponding event into the `EventMontior`. Document the linearization policy in your report. The rest of the test code is already present in the `task_4()` function of the lab template. You can run the code of `task_4()` using the command `./a.out 4`.

Hint: You might be able to reuse parts from task 2.

3 Submission Instructions

The submission in Studium should have two files:

1. Your report as a PDF document.

   All text should be written on a computer. Graphs can be drawn digitally or by hand.

   If you have any feedback or suggestions regarding the assignment, you're welcome to include them in your report as well. This will not affect the grading of your submission.

2. A ZIP file with your programmed solution.

   If you're using the provided template, please zip the entire `lab0*` directory.

Each student must create their own individual solution. You are allowed to discuss your work with others, but you shouldn't share code.