NOTE *This is a group assignment.*

START TIME *After you read the specified parts about synchronisation and deadlock you can start with this assignment.*

DURATION *Keep in mind the time you spend on this assignment: on average it will take **8 hours**.*

OBJECTIVES

- *gaining insight into concurrency issues;*
- *creating an algorithm using concurrency;*
- *getting hands on experience with threading interfaces;*
- *creating a simple benchmark for concurrent programs;*
- *understanding performance pitfalls when using concurrency.*

DEADLINE *You should hand it in before Thursday 14 October, 23:59. Send it to us using Brightspace.*

Specification

The goal of this assignment is to give an implementation of a simple (possibly bounded) buffer. A buffer stores data and some threads might be writing to it while others might be reading from it. As such, these buffers can be used to allow for communication between different threads, just like pipes. This data type can also be used for servers: the users add tasks to the buffer while the working threads read the tasks and execute them. For this reason, it is desirable to have an implementation of buffers that can be reused in a wide variety situations.

The standard operations on buffers are removing and adding an element to the buffer. The implementation should also allow for both bounded and unbounded buffers. The user of this data type should be able to set a bound or to let the buffer be unbounded. As such, there is a possibility that operations fail: elements could be removed from an empty buffer or something could be added to a buffer that is already full. For this reason, the success and failure of every operation must be logged.

In this assignment, the logger is represented by a vector of strings while the buffer is represented by a vector of integers. Even though it is desirable to have a polymorphic implementation so that it can be used for various data types, we restrict ourselves to integers for the sake of simplicity. For the logging service, you should implement the following operations:

- Write to the log.
- Read from the log.

For the buffer, you should implement the following operations:

- Add an integer to the buffer.
- Remove an integer from the buffer.

- Set a bound to the buffer.
- Make the buffer unbounded.

The success/failure of each of these operations must be logged. For example, a failure must be logged if an element is removed from an empty buffer or if an element is added while bound of the buffer is reached.

Method

The default interface in UNIX for threads is the POSIX API. This has many parts, including mutexes, condition variables, and shared memory. This interface is rather old, and therefore not really practical to work with. Instead, we are going to work with the modern C++ (version ≥ 11) interface. This interface is adaptable and simple to use. Below, the most important interfaces for this assignment are explained. This description is not complete.

NOTE *If you are familiar with the relevant POSIX thread interfaces, you can directly use those.*

The C++ API is adaptable and powerful. A good overview and more in depth explanations can be found on `https://www.cppreference.com`. Quickly scan the following pages, click on a name to directly go to the corresponding page.

- `std::thread::thread` (constructor of `std::thread`), to start a new thread.
- `std::mutex::lock` (method `lock()` of the class `std::mutex`). Acquire a lock on a mutex.
- `std::mutex::unlock` (method `unlock()` of the class `std::mutex`). Release a lock on a mutex.
- `std::vector::resize` (method `resize()` of class `std::vector`). Change the size of the vector (array). The size of the vector can be changed dynamically depending on the input. Note: if you resize the array to a larger capacity, the new values are initialised to default values. These could be different values that you expect and/or need.
- `std::stoi` (function `stoi`, converts a `std::string` to an integer). The usage is easy: `int number = stoi(arguments[1])`. However, it could be that the input is not a number (but text). If so, an exception called `std::invalid_argument` is thrown. To handle this exception a `try/catch` block is needed.

Problem

STEP 0 Download the new project in your current development environment of the shell assignment at `https://gitlab.science.ru.nl/operatingsystems/assignment2`.

STEP 1 Implement the logging operations.

STEP 2 Implement the operations on the buffer.

STEP 3 Mark the critical sections for the logging and the buffer using comments.

STEP 4 Argue that your implementation is free of deadlocks and starvation. If either of those is possible, try to fix it. However, if you ran out of time, you can also describe how one can improve upon your code.

STEP 5 Design tests to check the functional correctness of your implementation. You can consider tests that check whether your program gives the correct result if multiple threads are used and whether it works fine if there are multiple threads accessing and writer to the buffer.

HAND IN

- `buffer.cpp`, *with the comments added in step 3.*
- *A PDF file where you discuss the implementation. In the report, there should be a section in which the tests are discussed. You should also discuss the rational behind every test, and you should also analyze the edge cases.*
- *A section where you discuss the possibility of deadlocks/starvation in your implementation.*