**Параллельные структуры данных средствами библиотеки pthread**

**Общее описание**

В каждом варианте лабораторной работы требуется реализовать на языке C++ две параллельные структуры данных, используя инструменты библиотеки pthread (пользоваться примитивами синхронизации С++ 11 запрещено). Итоговая программа должна также содержать несколько тестовых сценариев - по одному для проверки каждой очереди в отдельности и тестовый сценарий, характерный для обеих структур данных, позволяющий проводить необходимые измерения времени работы потоков. Например, требуется реализовать очередь, работающую по принципу Multiple Consumers Single Producer (MCSP) (Multiple Readers Single Writer) и очередь Multiple Consumers Multiple Producer (Multiple Readers Multiple Writers), тогда реализуются сценарии MCSP и MCMP соответственно для проверки работы каждой, а общий тестовый сценарий будет запускать один поток, записывающий в очередь и множество потоков, читающих оттуда. Этот тест применяется к каждой очереди отдельно и производятся измерения времени работы каждого потока. Количество писателей, читателей, а также количество записей и чтений должны легко настраиваться (например, эти числа могут считываться из параметров командной строки или из конфигурационного файла).

**Сборка**

Все тесты должны собираться в один исполняемый файл утилитой make. Исходные файлы должны компилироваться с флагами -Wall и –Werror. Вызов make clean должен очищать все сгенерированные во время сборки файлы

**Сценарии тестирования**

В рамках данных тестов предполагается, что элементами тестируемой структуры являются целые числа. Результат работы каждого теста должен выводиться на экран, а в случае с измерением времени должна выводиться сравнительная таблица. В случае продолжительного (например, больше секунды) голодания потоков чтения нужно вызывать функцию *pthread\_yield*.

**Проверка писателей**

Запускается несколько потоков, которые одновременно записывают некоторые наборы значений в структуру. После завершения потоков проверяется, что структура содержит нужный набор значений.

**Проверка читателей**

Структура предварительно заполняется данными. Запускается несколько потоков, которые одновременно извлекают данные из структуры (в случае множества наборы распределяются между потоками заранее). После завершения потоков проверяется, что собранные данные вместе представляют собой значения, ранее располагавшиеся в структуре, а также тот факт, что структура пуста на момент окончания чтения (в случае множества).

**Общий тест**

Комбинация предыдущих методов. Для проверки используется массив, доступный всем читающим потокам. Длина массива 𝑛 равна количеству чисел, которыми заполняется тестируемая структура (целые числа 0 до 𝑛 − 1). Проверочный массив инициализируется нулями. При считывании элемента 𝑖, поток атомарно увеличивает значение 𝑖-го элемента массива на 1. В конце работы проверяется, что все значения в массиве равны 1. Должны проверяться все допустимые комбинации потоков чтения/записи, при условии, что их сумма не превышает количества потоков, которые могут быть одновременно запущены системой (например, 1 пишет, 1 читает; 2 пишут, 1 читает; и т.д.).

**Сравнение скорости работы**

Сценарии работы потоков те же, что и в предыдущих подразделах. В рамках каждого измерения должно гарантироваться, что наборы данных в каждом потоке одинаковы для обеих проверяемых структур. Для каждой конфигурации потоков должно производиться несколько запусков, после чего происходит усреднение времени работы. В случае множества необходимо производить два вида тестирования:

* *Рандомизированное*. В данном виде тестирования наборы данных генерируются случайным образом (но все еще единообразно для обеих структур).
* *Фиксированное*. Данные разбиваются таким образом, чтобы индексы потоков на одинаковых местах имели близкие значения. Например, общий размер множества - 6, два потока пытаются записывать 1, 3, 5 и 2, 4, 6, соответственно.

**Структуры данных**

* Множество (set) - набор элементов, не содержащий дубликатов. Можно реализовывать через. hashCode, как в книге, так и через компараторы, как в STL. Должны быть реализованы три метода:

1. *add(item)* - добавляет item в множество, возвращает true, если его ранее там не
2. было;
3. *remove(item)* - удаляет item из множества, возвращает true, если он там был;
4. *contains(item)* - возвращает true т. и тт., когда item является элементом множества;

* Очередь (queue). FIFO-контейнер, реализует три метода:

1. *enqueue(val)* - поместить элемент в конец очереди;
2. *dequeue()* - извлечь первый элемент очереди, если таковой имеется;
3. *empty()* - проверить, является ли очередь пустой;

* Стек (stack). LIFO-контейнер, реализует три метода:

1. *push(val)* - поместить элемент на вершину стека;
2. *pop(val)* - извлечь элемент с вершины стека, если таковой имеется;
3. *empty()* - проверить, является ли стек пустым;

**Виды параллельных структур данных**

1. Множество с “грубой” синхронизацией (Coarse-Grained Synchronization)[1, p.200]. Множество реализуется с помощью связного списка. "Грубая" синхронизация подразумевает взаимоисключающее блокирование всех методов - не более одного из них может выполняться в единицу времени.

2. Множество с “тонкой” синхронизацией (Fine-Grained Synchronization)[1, p.201]. Множество реализуется с помощью связного списка. "Тонкая" синхронизация подразумевает блокирование каждого узла вместо блокирования всего множества в "грубой".

3. Множество с “оптимистичной” синхронизацией (Optimistic Synchronization)[1, p.205]. Множество реализуется с помощью связного списка.

4. Множество с “ленивой” синхронизацией (Lazy Synchronization)[1, p.208]. Множество реализуется с помощью связного списка.

5. Очередь SCSP wait-free. Очередь, для которой известно, что лишь один поток может добавлять (enqueue) данные и лишь один поток может вынимать (dequeue) данные.

6. Очередь MCSP wait-free. Очередь, для которой известно, что лишь один поток может добавлять (enqueue) данные.

7. Очередь SCMP wait-free. Очередь, для которой известно, что лишь один поток может вынимать (dequeue) данные.

8. Очередь MCMP с блокировками. Полнофункциональная очередь с блокировками. [2, p. 226]

9. Стек с блокировками. [2, p. 226]

10. Стек без блокировок. [2, p. 246]

**Таблица заданий для различных вариантов**

|  |  |  |
| --- | --- | --- |
| **Номер варианта** | **Виды структур** | **Баллы** |
| 01 | 1, 2 | 7 |
| 02 | 1, 3 | 7 |
| 03 | 2, 3 | 8 |
| 04 | 1, 4 | 8 |
| 05 | 2, 4 | 9 |
| 06 | 3, 4 | 9 |
| 07 | 6, 8 | 12 |
| 08 | 7, 8 | 12 |
| 09 | 1, 5 | 8 |
| 10 | 2, 5 | 9 |
| 11 | 3, 6 | 10 |
| 12 | 2, 5 | 9 |
| 13 | 4, 5 | 10 |
| 14 | 2, 6 | 10 |
| 15 | 3, 6 | 10 |
| 16 | 4, 6 | 11 |
| 17 | 5, 6 | 11 |
| 18 | 5, 7 | 11 |
| 19 | 6, 9 | 14 |
| 20 | 5, 10 | 13 |

|  |  |
| --- | --- |
| **Виды структур** | **Баллы** |
| 1 | 3 |
| 2 | 4 |
| 3 | 4 |
| 4 | 5 |
| 5 | 5 |
| 6 | 6 |
| 7 | 6 |
| 8 | 6 |
| 9 | 8 |
| 10 | 8 |

Допускается реализация не всех структур. См. таблицу стоимости структуры. В случае нехватки балла для желаемой оценки допускается взять комбинацию из самых дорогих структур данных 9 и 10 на 16 баллов.

**Литература**

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

[2] Э. Уильямс, *Параллельное программирование на С++ в действии*, ДМК Пресс,2012

[3] https://gcc.gnu.org/onlinedocs/gcc-4.1.2/gcc/Atomic-Builtins.html