МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСИС»

Институт информационных технологий и компьютерных наук

Кафедра инженерной кибернетики

Курсовая работа

по дисциплине

«Объектно-ориентированное программирование»

на тему

«Библиотека ограниченной очереди без блокировок»

Выполнил:
студент 1-го курса,
гр. БПМ-22-3 Самсонов Н.О.

Проверил:
доцент, к.т.н. Полевой Д.В.

Москва, 2023

Оглавление

Описание задачи..... 3
Пользовательское описание..... 3
Техническое описание..... 6
Установка и сборка библиотеки..... 7
Тестирование..... 7
Список использованной литературы..... 8

Описание задачи

Необходимо разработать библиотеку LfQueue, предназначенную для реализации очереди без блокировки при доступе к элементам. Библиотека LfQueue предоставляет удобный интерфейс для добавления и извлечения элементов из очереди, обеспечивая безопасность при многопоточном доступе. Она основана на алгоритме lock-free (без блокировки) для максимальной эффективности и производительности.

Библиотека предоставляет следующие возможности:

1. Добавление элементов в очередь: метод `push` позволяет добавить элемент в конец очереди.
2. Извлечение элементов из очереди: метод `pop` позволяет извлечь элемент из начала очереди.
3. Проверка наличия элементов в очереди: метод `is_empty` возвращает true, если очередь пуста, и false в противном случае.
4. Дополнительные методы `produce/consume` для производства/потребления задач.
5. Задание параметров в произвольном порядке с помощью аргументов командной строки.
6. Безопасность при многопоточном доступе: библиотека реализована с использованием lock-free алгоритма, что позволяет обеспечить безопасность при одновременном доступе из нескольких потоков.

Пользовательское описание

Библиотека LfQueue предоставляет простой и эффективный способ работы с очередью для многопоточных программ. Для начала работы с библиотекой, необходимо установить исходники (раздел «Установка и сборка библиотеки»), подключить её к вашему проекту (`#include <lf_queue/lf_queue.hpp>`) и использовать пространство имён `lockFreeQueue, parser` и `lockFreeQueueProcessor`.

Основные шаги для работы с библиотекой LfQueue (многопоточное использование):

1. Создание очереди:
   Создайте объект класса LfQueue, указав тип элементов в угловых скобках, а также размер буфера очереди в конструкторе.
   Например: `LfQueue<int> queue(128);`

2. Создание потока(ов) для производства задач:
   Создайте и инициализируйте поток, используя функцию `produce`, передавая ссылку на очередь в качестве аргумента и количество времени на производство/потребление задачи.
   Например: `std::thread th_prod{produce, std::ref(queue), 0};`

3. Создание потока(ов) для потребления задач:

Создайте и инициализируйте поток, используя функцию `consume`, передавая ссылку на очередь в качестве аргумента и количество времени на производство/потребление задачи.

Например: `std::thread th_cons{consume, std::ref(queue), 0};`

4. Дождитесь выполнение потоков:
   Для корректного исполнения программы нужно вызвать метод join для каждого созданного потока.
   Например: `th_prod.join(); th_cons.join();`

5. Получение размера очереди:
   Используйте метод `is_empty()`, который возвращает true, если очередь пуста, и false, если в очереди есть элементы.
   Например: `std::cout << queue.is_empty() << std::endl;`

   Наглядный пример использования библиотеки можно найти в папке prj.cw/prj.test/example.cpp.

6. Замер времени выполнения:
   Для того, чтобы узнать, сколько по времени заняло производство/потребление всех задач, можно воспользоваться аргументом командной строки `--time` при запуске исполняемого файла.
   Например: `./example --time.`

7. Запуск программы:
   При запуске исполняемого файла вы можете указать аргументы командной строки для того, чтобы задать: количество задач, размер буфера очереди, время для потребления одной задачи, время для производства одной задачи, количество производственных потоков или количество потребительных потоков (значения по умолчанию для каждого из них соответсвенно: 1000, 4096, 0, 0, 1, 1). Название этих параметров можно узнать при помощи аргумента `help` (либо просто запустив исполняемый файл без аргументов: `./example`).
   Например: `./example --ntasks 100000 --bufsize 1024 --constime 0 --prodtime 0 --nprodthreads 4 --nconsthreads 4.`
   Или: `./example --help.`

Основное отличие ограниченной очереди без блокировок от очереди, написанной при помощи взаимно исключающей синхронизации (mutex) заключается в том, что при работе с большим количеством потоков класс LfQueue обеспечивает достаточно надежную синхронизацию между ними (максимальное количество потерянных задач - 1). Однако преимущества класса LfQueue в том, чтобы обрабатывать большое количество задач, время исполнения которых стремится к нулю (например, обработчик прерываний). В случаях, когда время на одну задачу превышает несколько миллисекунд, прироста производительности не будет заметно.

При сравнении очереди, написанной на mutex и ограниченной очереди без блокировок для 100000 задач с временем выполнения 0 миллисекунд, можно заметить, что

lock-free очереди существенно выигрывают в производительности. Ниже предоставлен график зависимости время выполнения всех задач для lock-free очереди (синяя кривая) и для очереди, написанной на mutex (красная кривая).

Рисунок 1 - сравнение времени выполнения задач mutex/lock-free очередей

Вот так выглядит график зависимости выполнения задач с теми же параметрами для lock-free очереди:

Рисунок 2 - сравнение времени выполнения задач lock-free очереди

Методика получения данных для графиков функций:

1. Получение данных о времени исполнения для mutex очереди выполнялись непосредственной подстановкой значений в заранее написанную очередь, основанную исключительно на блокировках, которая содержит только методы и функции, содержащиеся в библиотеке LfQueue. Для демонстрации преимущества очереди без блокировок, приведены результаты замеров при суммарном количестве потоков, равным 12 (*рисунок 1*).
2. Получение данных о времени исполнения для lock-free очереди производилось по той же самой методике, за исключением того, что некоторые параметры передавались при помощи аргументов командной строки (см. раздел “Тестирование”). График, изображенный на *рисунке 2* ограничен по оси X до ~140 для того, чтобы продемонстрировать разницу между суммарным количеством продьюсеров и консьюмеров, выполнивших поставленную задачу за примерно то же самое время, между lock-free очередью и mutex очередью.
3. Получение данных проводилось посредством методики, описанной в п.1 и п.2 на операционной системе Linux ubuntu 6.2.0-23-generic (x86_64). Дополнительную информацию о характеристиках процессора можно найти в репозитории github [1] в файле cpuinfo.txt.

Техническое описание

Классы:

`LfQueue` - класс, представляющий собой ограниченную очередь.

Документация по классам, сгенерированная с помощью DoxyGen:

Шаблон класса lockFreeQueue::LfQueue< T >

Открытые члены

LfQueue (unsigned BufSize)
*Конструктор **LfQueue.***
Исключения
std::runtime_error, размер буфера должен быть степенью 2

~LfQueue ()=default
Деструктор

bool **push** (T data)
Добавление элемента в конец буфера

bool **pop** (T &data)
Удаление элемента из начала буфера

bool **is_empty** () const
Проверка пустоты очереди

Установка и сборка библиотеки

Файлы и документация расположены по ссылке на репозиторий github [1]. Оттуда необходимо клонировать репозиторий командой

```
git clone https://github.com/we11cIJm/secondSemester.git
```

Также, перед использованием библиотеки (если вы собираетесь активировать мод тестирования), необходимо установить GTest из vcpkg командой `./vcpkg install gtest` (укажите сразу после слова “gtest” ваш триплет через двоеточие, например `./vcpkg install gtest:x64-linux`).

После этого необходимо собрать проект с помощью CMake (версия должна быть 3.14 или выше). Сборка осуществляется вручную, следуя данному алгоритму:

1. Перейдите в директорию secondSemester/samsonov_n_o.
2. Откройте командную строку и напишите команду “`cmake -B build -DCMAKE_TOOLCHAIN_FILE=<путь до vcpkg.cmake> && cmake --build build --config Release && cmake --install build --component test --prefix <путь установки>`” (или команда `cmake --install build --component --prefix <путь установки>` установит исходники, необходимые для работы с библиотекой в указанную папку). Для того, чтобы активировать возможность тестирования корректности работы библиотеки после `cmake -B build` добавьте `-DTESTING_MODE=ON`.
3. Документация будет сгенерирована в директорию docs.

Тестирование

После сборки вы можете протестировать пример использования библиотеки, а также исполняемый файл lf_queue.test, в котором использована библиотека gtest. Для этого вам нужно будет выполнить следующие действия из командной строки:

Тестирование корректности работы библиотеки:

1. Перейти в папку инсталяции библиотеки, которую указали после `--prefix` в предыдущем разделе. Затем в папку bin.
2. В командной строке текущей директории напишите команду `./example --bufsize 2048` (должно быть написано “everything’s good”) или `./example --bufsize 2049` (программа должна экстренно завершиться в связи с несоблюдением контракта по установлению размера буфера) для того, чтобы проверить корректность работы библиотеки.
3. Убедитесь в том, что запуск с разными аргументами командной строки пройдены успешно и не занимают большого количества времени.
4. Также можете сделать замеры при разных параметрах, запустив исполняемый файл командой `./example --ntasks 1000 --bufsize 4096 --constime 0 --prodtime 0 --nprodthreads 4 --nconsthreads 4`. Результат выполнения можно получить, добавив аргумент `--time`.

5. Для того, чтобы проверить приблизительное совпадение полученных данных вы можете запустить исполняемый файл с примером таким образом:

   `./example --time --ntasks 100000 --bufsize 128 --nprodthreads N --nconsthreads N` (вместо N подставьте любое значение). Например, при подстановке N=64 вы должны получить время (после исполнения программы появится строка “Total duration: <ваше время> ms”), примерно равное 1000 мс (погрешность зависит от машины, на которой производятся замеры). После получения результатов, сравните распределение значений с графиком (*рисунок 2*).

Тестирование в “моде тестирования”:

1. Совершите повторную сборку библиотеки (раздел “Установка и сборка библиотеки”), удалив папку build в папке samsonov_n_o. При сборке библиотеки добавьте после команды `cmake -B build` опцию `-DTESTING_MODE=ON`.
2. Перейти в папку инсталяции библиотеки, которую указали после `--prefix` в предыдущем разделе. Затем в папку bin.
3. В командной строке текущей директории напишите команду `./lf_queue.test` для того, чтобы убедиться, что библиотека проходит тесты. Также можно использовать команду (если для вашей системы она доступна) `while ./lf_queue.test; do; done` для того, чтобы убедиться, что подавляющее большинство тестов проходят и изредка очередь теряет 1 задачу в одном из многочисленно пройденных тестов.

Список использованной литературы

1. Репозиторий GitHub - URL: https://github.com/we11cIJm/secondSemester.git (дата обращения 12.06.2023).

2. Anthony Williams, C++ Concurrency in Action: Practical Multithreading, 2012.

3. Herb Sutter, Lock Free Programming or Juggling Razor Blades, 2014.

4. Hans Boehm "Using weakly ordered C++ atomics correctly", 2016.