Санкт-Петербургский политехнический университет Петра Великого

Институт машиностроения, материалов и транспорта

Кафедра «Мехатроника и роботостроение (при ЦНИИ РТК)”»

**Курсовой проект**

по дисциплине «Объектно-ориентированное программирование»

«Потокобезопасная очередь без блокировок»

Студент гр. 3331506/00401 Земский С. А.

Преподаватель Ананьевский М. С.

« »\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

Санкт-Петербург

2023 г.

# Введение

В настоящее время многопоточность стала неотъемлемой частью разработки программного обеспечения. Она позволяет увеличить производительность приложений и обеспечить более эффективное использование ресурсов компьютера. Однако, при работе с многопоточностью необходимо учитывать возможность одновременного доступа к общим ресурсам из разных потоков, что может привести к проблемам синхронизации и блокировок.

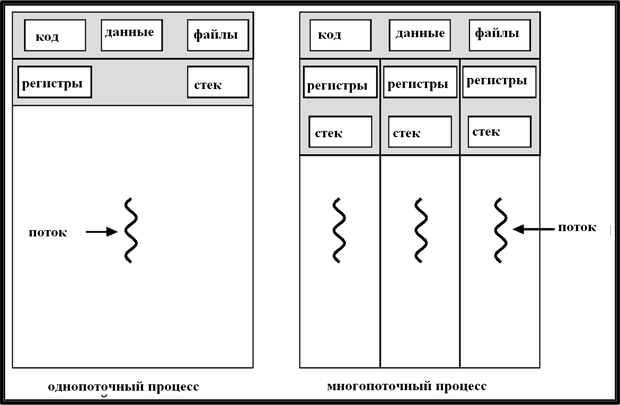


Рисунок 1 – Принципы работы однопоточного/многопоточного проекта

Одной из наиболее распространенных проблем при работе с многопоточностью является проблема блокировок. Блокировки возникают в тех случаях, когда один поток пытается получить доступ к ресурсу, который уже занят другим потоком. В результате этого поток, который ждет освобождения ресурса, блокируется и не может продолжить свою работу до тех пор, пока ресурс не будет освобожден.

Для решения проблемы блокировок и обеспечения безопасной записи и чтения из разных потоков была разработана очередь без блокировок на языке программирования С++. Очередь без блокировок позволяет избежать блокировок и увеличить производительность приложений, работающих с многопоточностью.

Цель данного курсового проекта - разработать и реализовать очередь без блокировок на языке программирования С++, которая будет обеспечивать безопасную запись и чтение из разных потоков. В рамках работы будет проведен анализ существующих решений, разработана архитектура очереди без блокировок, реализованы необходимые методы и проведены тесты на производительность и безопасность работы очереди.

# Технические требования

Реализовать простую шаблонную циклическую очередь, безопасную для одновременной записи и чтения из двух потоков (один поток читает, другой пишет), не использующей механизмы взаимной блокировки потоков. **Требования:**

· Память для хранения данных аллоцируется статически внутри очереди.

· Два шаблонных параметра: T – тип данных, CAPACITY – ёмкость очереди (максимальное количество хранимых элементов типа T)

· Очередь должна поддерживать конструктор копирования и оператор присваивания

**Важные замечания, возникшие во время написания программы:**

·Т. к. очередь предназначена для работы с шаблонами всё описание класса было в header-based библиотеке.

· Для того, чтобы очередь могла использоваться для произвольных типов данных нельзя было использовать std::atomic<T>, который сильно ограничивает возможные типы данных. Для решения данной задачи использовалась библиотека [std::aligned\_storage](https://en.cppreference.com/w/cpp/types/aligned_storage). С её помощью шаблонный тип Т приводился к тривиальному типом стандартной компоновки, подходящим для использования в качестве неинициализированного хранилища для любого объекта.

· Из-за большой сложности обработки исключений они не использовались.

· Безопасность обеспечивалась не использованием блокировок(mutex-ов), а тем, что методы добавления элемента (push) и извлечения (pop) модифицировали только указатели на конец (rear) и начало (front) очереди соответственно.

·

# Синтаксис программы

·Библиотека <queue.h> включает в себя описание класса Queue, который содержит :

Таблица 1 — Описание класса

|  |  |
| --- | --- |
| 1 Данные: | Указатели на первый, последний элемент, массив данных T, флаг пустоты очереди. |
| 2 Конструкторы | Конструктор по умолчанию, копирования, деструктор |
| 3 Операторы | Операторы присваивания и обращения по интдексу |
| 4 Функции | Вставки (push) и извлечения (pop) элемента. |

Для анализа целесообразности использования многопоточности была создана соответствующая программа, использующая описанную библиотеку. Результаты исследования представлены в следующем разделе.

# **Сравнение результатов**

В этом разделе будет рассмотрено сравнение результатов работы программы, использующей потокобезопасную очередь в однопоточном и многопоточном режимах. Для этого в создаётся очередь на 500000 элементов, которые параллельно добавляются/ извлекаются.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 2 — Многопоточное использование

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 3 — Однопоточное использование

Как видно, по выводу из консоли программа в многопоточном режиме работает более, чем в 2 раза быстрее.

# Список использованной литературы

1. <https://en.cppreference.com/w/cpp/types/aligned_storage>
2. <https://habr.com/ru/articles/219201/>

# Приложение

Приложение А

queue3.h

#ifndef QUEUE3\_QUEUE3\_H  
#define QUEUE3\_QUEUE3\_H  
  
#include <iostream>  
#include <cstring>  
#include <type\_traits>  
  
  
template<typename T, const unsigned int *CAPACITY*>  
class Queue {  
private:  
 long front;  
 long rear;  
 std::aligned\_storage\_t<sizeof(T), alignof(T)> queue[*CAPACITY*];  
 long temp\_front;  
 T output\_val;  
  
public:  
 bool empty ;  
  
public:  
 Queue() : front(0), rear(0), empty(true) {} ;  
 Queue(const Queue &other) ;  
 ~Queue() = default;  
  
public:  
 Queue& operator=(const Queue &other);  
  
private: // private for impossibility user's "breaking" structure of class  
 const T& operator[](unsigned int pos) const {return \*std::launder(reinterpret\_cast<const T\*>(&queue[pos]));}  
  
public:  
 bool push(const T& new\_val);  
 T pop();  
};  
  
  
template<typename T, const unsigned int *CAPACITY*>  
Queue<T, *CAPACITY*>::Queue(const Queue &other) {  
 for (int index = 0; index < *CAPACITY*; index++) {  
 ::new(static\_cast<std::aligned\_storage\_t<sizeof(T), alignof(T)>\*> (&queue[index]))  
 T(\*std::launder(reinterpret\_cast<const T\*>(&other.queue[index])));  
 }  
 front = other.front;  
 rear = other.rear;  
 empty = other.empty;  
}  
  
template<typename T, const unsigned int *CAPACITY*>  
Queue<T, *CAPACITY*>& Queue<T, *CAPACITY*>::operator=(const Queue &other) {  
 for (int index = 0; index < *CAPACITY*; index++) {  
 ::new(static\_cast<std::aligned\_storage\_t<sizeof(T), alignof(T)>\*> (&queue[index]))  
 T(\*std::launder(reinterpret\_cast<const T\*>(&other.queue[index])));  
 }  
 front = other.front;  
 rear = other.rear;  
 empty = other.empty;  
 return \*this;  
}  
  
template<typename T, const unsigned int *CAPACITY*>  
bool Queue<T, *CAPACITY*>::push(const T &new\_val) {  
 if (front == (rear + 1)% *CAPACITY*) return false;  
 ::new(static\_cast<std::aligned\_storage\_t<sizeof(T), alignof(T)>\*>  
 (&queue[(front == rear) && empty? rear: (rear + 1) % *CAPACITY*])) T(new\_val);  
 rear = ((front == rear) && empty) ? rear: (rear + 1) % *CAPACITY*;  
 empty = false;  
 return true;  
}  
  
template<typename T, const unsigned int *CAPACITY*>  
T Queue<T, *CAPACITY*>::pop() {  
 if (empty) return T();  
 temp\_front = front;  
 if (front == rear) empty = true;  
 else front = (front + 1) % *CAPACITY*;  
 ::new(&output\_val) T(\*std::launder(reinterpret\_cast<const T\*>(&queue[temp\_front])));  
 memset(&queue[temp\_front], 0, sizeof(T));  
 return output\_val;  
}  
  
#endif //QUEUE3\_QUEUE3\_H

Приложение B

main.cpp

#include <iostream>  
#include "queue3.h"  
#include <thread>  
#include <ctime>  
  
  
int main()  
{  
 const unsigned int N = 500000;  
 Queue<unsigned int, N> l;  
 unsigned int start\_time = clock();  
 /\*std::thread push\_thread([&] () {  
 for (unsigned int count = 0; count < N \* 100; count++){  
 l.push(count);  
 }  
 });  
 std::thread pop\_thread([&] () {  
 for (unsigned int count = 0; count < N \* 100; count++){  
 l.pop();  
 }  
 });  
 push\_thread.join();  
 pop\_thread.join();\*/  
 for (unsigned int count = 0; count < N \* 100; count++) l.push(count);  
 for (unsigned int count = 0; count < N \* 100; count++) l.pop();  
 unsigned int end\_time = clock();  
 std::cout <<"working time = " << end\_time - start\_time << " ms.";  
 return 0;  
}