# Практическое занятие Программирование без блокировок

## Задание 1. std::future<>, std::promise<>

Реализуйте взаимодействие потоков:

* Дано: vector<int> v;
* Запустите параллельно два (несколько) **detach** - ed (независимых) потока. Каждый поток будет добавлять в вектор заданное количество значений (например: первый - “N” элементов со значением «1, 2, 3…» и второй - “M” элементов со значением «-1, -2…»).
  + Обеспечьте защиту вектора при его модификации
  + Обеспечьте формирование признака окончания операции (завершения действия)
  + Подумайте, **какие** параметры нужно передать для потоковой функции
  + Подумайте, **как** следует/можно формировать параметры для потоковой функции
* Запустите еще один поток, который должен окончательно сформированный предыдущими потоками вектор преобразовать согласно заданному Вами условию
  + В зависимости от условий запуска потока – его можно сделать независимым или дождаться его завершения

## Задание 2. Реализация spinlock. std::atomic\_flag

1. Создайте класс таким образом, чтобы его можно было использовать в качестве мьютекса, но без блокировки потока. Реализуйте методы lock(), unlock() и try\_lock()  
   При реализации требуется предусмотреть максимально возможную производительность => подумать над заданием соответствующего упорядочения
2. Проверьте использование разработанного класса в обертках типа lock\_guard или unique\_guard
3. Для проверки работоспособности замените мьютекс в реализации потокобезопасного стека (или очереди) из предыдущего задания на разработанный класс
4. Оцените (качественно) полученный выигрыш на одном и том же наборе данных при использовании обычного мьютекса и разработанного spinlock-а

## Задание 3. Атомарные типы данных.

Разработайте класс:

Template<typename T> class A{

T m\_a; //обычное данное

… //требуемые данные и методы

};

, который должен «считать» текущее количество объектов типа A в программе.

Если понадобится защита от многопоточной модификации разделяемых данных, реализуйте три разных варианта защиты:

* Мьютексом
* Разработанным в предыдущем задании spin lock-ом
* Использованием атомарных данных

Попробуйте качественно оценить эффективность при выполнении программы в примерно одинаковых условиях, но с разными способами защиты

# Задание 4. Потокобезопасная кольцевая очередь фиксированного размера на базе условных переменных. std::condition\_variable::wait\_for()

Имитируем формирование и обработку «заявок» посредством многопоточной очереди. Для каждой операции (и чтения, и записи) запускаем отдельный поток (потоки запускаются в любом порядке). Если число читателей совпадает с числом писателей, то задача решается легко, Но! Мы не ищем простых путей => усугубляем ситуацию!!! => Существенно! - читателей и писателей может быть разное количество!

Каждый поток-писатель должен попытаться занести свою «заявку» в конец очереди и уведомить одного читателя о том, что появилась заявка => можно обрабатывать. При этом!

если в очереди нет места, писатель должен ждать, пока место появится,

или работа очереди была прекращена, поток тоже должен завершиться

или если читателей нет, и в течение фиксированного периода ни один читатель так и не появился, поток тоже нужно завершить.

Каждый поток-читатель должны изъять заявку из начала очереди и обработать. При этом!

если в очереди заявок нет, читатель должен ждать, пока какой-нибудь писатель заявку сформирует

или работа очереди была прекращена, , поток тоже должен завершиться

или если писателей нет, и в течение фиксированного периода ни один писатель так и не появился, , поток тоже нужно завершить

• Требуется реализовать класс

class Queue{

static const size\_t size; // размер буфера (альтернатива – можно задать целочисленным параметром шаблона)

char buf[size]; // будет использоваться как кольцевой буфер. Для упрощения - «заявка» - это символ. Для дальнейшей визуализации пусть писатели заносят в буфер код символа в верхнем регистре, а для читателей пусть «обработка заявки» заключается в выводе считанного символа в нижнем регистре

… //другие переменные класса для организации работы с кольцевым буфером

…// мьютекс для синхронизации и читателей, и писателей

…//условная переменная для сигнализации о том, что в очередь поставлена новая заявка => можно обрабатывать

…// условная переменная для сигнализации о том, что в очереди освободилось место => можно заносить новую заявку

…// признак – работу всех потоков нужно остановить (возможно эта переменная должна быть атомарной?)

• Классическое копирование и присваивание очереди смысла не имеет => ???

• Можно попробовать для потокобезопасной очереди реализовать операции перемещающего копирования и присваивания

• деструктор? – что делать с потоками, которые ждут освобождения места в очереди или появления заявки?