# Практическое задание по теме: «Параллельное программирование с блокировками»

## once\_flag, call\_once, mutex, async, future, conditional\_variable

## Задание 1. Асинхронный запуск потоков. Получение отложенного результата. Параллельная реализация std::accumulate()

Задача: сформировать среднее значение для контейнеров (vector, deque) с элементами любого типа (int, float,…)

1. Создайте и заполните значениями контейнер с элементами типа int (количество элементов и значения можно формировать случайным образом).  
   Замечание: количество элементов должно быть достаточно большим, чтобы оценить результаты распараллеливания задачи
2. Измерьте время работы std::accumulate() для суммирования элементов от begin() до end()
3. Решите ту же задачу посредством параллельного суммирования:
   1. Исходя из количества вычислительных ядер, запустите посредством **std::async()** разумное количество потоков, каждый из которых должен получать сумму заданного диапазона
   2. Полученные частичные результаты по мере готовности требуется тоже просуммировать
   3. Измерьте общее время
   4. Попробуйте задать количество потоков   
      меньше, чем количество вычислительных ядер,   
      больше чем количество вычислительных ядер
4. Сравните полученные результаты

## Задание 2. Обработка исключений посредством future

Реализуйте в отдельном дочернем потоке прием целых значений от пользователя.

Если пользователь вводит НЕ целое, должно быть сгенерировано исключение.

Обработку исключения требуется реализовать в родительском потоке.

Подсказка: можно использовать функцию std::**stoi**() – С++11, которая генерирует исключения:

* **std::invalid\_argument**, если преобразование не может быть выполнено
* **std::out\_of\_range**, если преобразованное значение будет выходить за границы диапазона значений типа результата

# Задание 3. Пинг-понг на условных переменных

В заголовке все сказано…

# Задание 4. Взаимодействие посредством std::promise<>, std::shared\_future<>

Требуется организовать следующее взаимодействие потоков:

Поток №1: должен создать **динамический массив** заданного размера и заполнить его значениями. : после того, как данные сформированы, потоки 2, 3, и 4 должны **параллельно** выполнить каждый свою работу, а именно:

Поток № 2 – просуммировать значения

Поток № 3 – найти минимальное значение

Поток № 4 - найти максимальное значение

Все результаты требуется получить и вывести в родительском потоке.

Подумайте – где/когда нужно освободить память.

# Задание 5. Пул потоков. std::thread, std::mutex, std::conditional\_variable

Разработаем простой пул потоков, который позволяет:

* создавать разумное количество потоков
* и многократно их использовать, пока существует пул
* при уничтожении пула все потоки должны завершиться.

Ограничения (упрощения) нашей реализации:

* **не** будем генерировать и обрабатывать исключения
* все задания имеют одну и ту же сигнатуру
* количество заданий может быть любым
* время жизни пула формируется по общим правилам (при вызове деструктора все потоки должны завершиться).
* Подумайте, как можно уведомить все потоки, что они могут завершиться?
* можно предусмотреть остановку работы пула извне
* чтобы можно было хоть как-то попробовать измерить время работы, задания будем добавлять в пул все сразу и централизовано в функции main (в жизни обычно задания формируются разными клиентами в произвольном месте, в любое время и в произвольном порядке)
* при добавлении нового задания в пул уведомляем один поток, что он может взять задание на обработку
* Подсказка: так как устройство вывода будет разделяться разными потоками, что нужно сделать, если в каждом задании будет осуществляться вывод на консоль?
* Интересно было бы «посчитать» сколько заданий выполнил каждый поток

Было бы очень интересно измерить и сравнить время выполнения задачи при реализации посредством пула потоков и при запуске для каждого задания своего потока

## Задание 6. Потокобезопасная кольцевая очередь фиксированного размера на базе условных переменных. std::condition\_variable::wait\_for(),std::cv\_status

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?