АВС. ИПР 2.

Приемы разработки многопоточного кода.

Цель: изучить приемы организации работы многопоточных программ.

Общие указания. В работе нужно использовать язык C++, стандарт C++11 и выше. Необходимо использовать модули стандартной библиотеки: thread, mutex, atomic, future, conditional_variable. Лушче всего каждую задачу и каждый пункт каждой задачи реализовывать отдельной функцией или модулем одной программы.

Часть 1. Параллельный обход массива элементов.

Пусть нам дан массив из NumTasks однотипных элементом которые надо обработать NumThreads потоками. Если сложность обработки всех элементов приблизительно одинакова, то можно разбить массив на NumThreads частей состоящих из примерно NumTasks/NumThreads элементов и обработать каждую часть последовательно в отдельном треде. Однако, если время работы сильно зависит от значения элемента массива или случайно, то статическое планирование может оказаться невыгодным. Пример: чтение сотни тысяч файлов с диска размеров от 4Кб до 1Мб. В данном случае мы можем использовать общий потокобезопасный счетчик, значение которого есть индекс первого необработанного элемента. Каждый поток запрашивает индекс следующего необработанного элемента и увеличивает счетчик на 1. Если полученный индекс меньше NumTasks, поток запускает обработку указанного элемента, иначе завершает свою работу.

Задача. Написать функцию, которая инстанциирует массив из NumTasks байт, в каждом из которых записан 0. Запустить NumThreads потоков, каждый из которых читает потокобезопасный индекс, увеличивает его на один и прибавляет единицу к элементу массива по этому индексу. Засечь время работы всех потоков. После завершения работы всех потоков, необходимо проверить корректность заполнения массива и вывести на экран время работы.

1. Реализовать потокобезопасный счетчик двумя способами: блокирующий при помощи std::mutex и неблокирующий при помощи std::atomic
2. Измерить время работы функции, использующей разные реализации счетчика при NumTasks=1024*1024 и NumThreads={4, 8, 16, 32}.

3. Добавьте усыпление потока на 10ns после каждого инкремента элемента массива. Повторите измерения из пункта 2.

Часть 2. Производитель-потребитель.

Производитель-потребитель является распространенным шаблоном проектирования с множеством вариантов реализации. Обычно, производиетль и потребитель связываются потокобезопасной очередью задач, в которую пишет один или несколько производителей и читает один или несколько потребителей. В этой части необходимо разработать несколько реализаций потокобезопасной очереди сообщений.

Задача. Написать код инстанциирующий ConsumerNum потребителей и ProducerNum производителей. Связать их потокобезопасной очередью, хранящей однобайтные элементы. Потребители инстанциируют локальный счетчик, после чего начинают читать очередь и прибавлять вычитанные значения к локальному счетчику. Каждый производитель записывает TaskNum единиц в очередь и завершает свою работу. Когда все производители записали свои сообщения и потребители опустошили очередь, потребители завершают свою работу и возвращают получившиеся суммы. Необходимо засечь время работы производителей и потребителей и проверить, что итоговая сумма результатов всех потребителей равна ProducerNum / TaskNum. Протестировать для ProducerNum={1, 2, 4}, ConsumerNum={1, 2, 4}, TaskNum=4*1024*1024. Интерфейс очереди:

```
class queue
{
public:
    // Записывает элемент в очередь.
    // Гсли очередь фиксированного размер и заполнена,
    // поток повисает внутри функции пока не освободится место
    void push(uint8_t val);
    // Если очередь пуста, ждем 1 мс записи в очередь.
    // Если очередь не пуста, помещает значение головы в val,
    // удаляет голову и возвращает true.
    // Если очередь по прежнему пуста, возвращаем false
    bool pop(uint8_t& val);
};
```

Задачу необходимо решить для следующий вариантов реализации очереди:

1. Динамическая очередь с использованием std-контейнеров и std::mutex.

2. Очередь фиксированного размера QueueSize с использованием std::mutex и std::condition_variable без busy wait. Протестировать для QueueSize={1, 4, 16}.
3. Очередь фиксированного размера с использованием std::atomic. Используйте std::conditinal_variable и std::mutex только чтобы оповещать потоки об освободившихся ячейках или новых задачах Протестировать для QueueSize={1, 4, 16}.

Вопросы к сдаче.

1. Примитивы синхронизации: мониторы, семафоры, мьютексы.
2. Варианты реализации мьютексов в системе.
3. Варианты реализации atomic.
4. Атомарные операции.