Задачи

1 блок

1. Реализовать указанный параллельный алгоритм (назначается преподавателем) с применением MPI.NET:

- чёт-нечётная сортировка;
- быстрая сортировка;
- быстрая сортировка с использованием регулярного набора образцов; • алгоритм Флойда; • алгоритм Прима.

Исходные массивы и графы хранятся в файлах. Результаты работы также записываются в файл. Примеры таких файлов приложены в соответствующих папках в Git. Все числа в файлах представлены в ASCII в десятичной системе счисления.

Размер массива для сортировки – не менее 1000000 элементов. В файл массив записывается в одну строку с пробелом между элементами.

Число вершин V в графе не менее 5000, число рёбер – не менее 1000000 и не более V2/2. В файле с исходным графом данные представлены следующим образом:

- 1 строка – число вершин в графе;
- Последующие строки – описание рёбер в виде троек {индекс_вершины индекс_вершины вес_ребра}. Индекс первой вершины строго меньше индекса второй вершины.

Результат алгоритма Прима – файл с числом вершин в первой строке и весом полученного остовного дерева во второй. Результат алгоритма Флойда – записанная в файл построчно с пробелом между элементами матрица путей, где для каждого элемента индекс строки – начальная вершина пути, индекс столбца – конечная вершина пути.

Предусмотреть корректное завершение работы отдельных процессов.

2 блок

2. Реализовать решение упрощённой задачи «производитель-потребитель» (буфер не имеет верхней границы) с указанным преподавателем средством синхронизации (атомарные операции, мьютексы, семафоры, мониторы):

• Производители – объекты, кладущие некоторые объекты (например, числа, строки или более сложные объекты-заявки) нестатическими методами в экземпляр класса List<T>;

• Потребители – объекты, извлекающие заявки из экземпляра List<T> в нестатических методах;

• Между двумя последовательными добавлениями у одного и того же производителя или двумя последовательными изъятиями у одного и того же потребителя вставляется пауза (например, с помощью Thread.Sleep);

• Количество производителей и потребителей задаётся константами;

• При запуске программы создаются производители и потребители. Они прекращают работу по нажатию произвольной клавиши. При этом завершение работы производителей и потребителей должно быть корректно реализовано (Thread.Kill таковым не является).

3. Реализовать объект ThreadPool, реализующий паттерн «пул потоков» с поддержкой Continuation (наподобие https://docs.microsoft.com/en-us/dotnet/api/system.threading.threadpool?view=netframework-4.8 + https://docs.microsoft.com/enus/dotnet/api/system.threading.tasks.taskfactory?view=netframework-4.8).

Пул потоков:

- Число потоков задаётся константой в классе пула или параметром конструктора.

- У каждого потока есть два состояния: ожидание задачи, выполнение задачи

- Задача — вычисление некоторого значения, описывается в виде Func<TResult> и инкапсулируется в объектах интерфейса IMyTask<TResult>

- Добавление задачи осуществляется с помощью нестатического метода класса пула Enqueue(IMyTask<TResult> a).

- При добавлении задачи, если в пуле есть ожидающий поток, то он должен приступить к ее исполнению. Иначе задача будет ожидать исполнения, пока не освободится какой-нибудь поток.

- Класс должен быть унаследован от интерфейса IDisposable и корректно освобождать ресурсы при вызове метода Dispose().

- Метод Dispose должен завершить работу потоков. Завершение работы коллаборативное, с использованием CancellationToken — уже запущенные задачи не прерываются, но новые задачи не принимаются на исполнение потоками из пула. Возможны два варианта решения --- дать всем задачам, которые уже попали в очередь, досчитаться, либо выбросить исключение во все ожидающие завершения задачи потоки.

- Предусмотреть конфигурирование пула, чтобы была возможна как реализация стратегии Work Stealing, так и Work Sharing.

IMyTask:

- Свойство IsCompleted возвращает true, если задача выполнена

- Свойство Result возвращает результат выполнения задачи

- В случае, если соответствующая задаче функция завершилась с исключением, этот метод должен завершиться с исключением AggregateException, содержащим внутри себя исключение, вызвавшее проблему

- Если результат еще не вычислен, метод ожидает его и возвращает полученное значение, блокируя вызвавший его поток

- Метод ContinueWith — принимает объект типа Func<TResult, TNewResult>, который может быть применен к результату данной задачи X и возвращает новую задачу Y, принятую к исполнению

- Новая задача будет исполнена не ранее, чем завершится исходная

- В качестве аргумента объекту Func будет передан результат исходной задачи, и все Y должны исполняться на общих основаниях (т.е. должны разделяться между потоками пула)

- Метод ContinueWith может быть вызван несколько раз

- Метод ContinueWith не должен блокировать работу потока, если результат задачи X ещё не вычислен

- ContinueWith должен быть согласован с Shutdown --- принятая как ContinueWith задача должна либо досчитаться, либо бросить исключение ожидающему её потоку.

Ограничения:

- В данной работе запрещено использование TPL, PLINQ и библиотечных классов Task и ThreadPool.

- Все интерфейсные методы должны быть потокобезопасны

- Для каждого базового сценария использования должен быть написан несложный тест (добавление 1 задачи, добавление задач, количественно больших числа потоков, проверка работы ContinueWith для нескольких задач). Для всех тестов обязательна остановка пула потоков.

- Также должен быть написан тест, проверяющий, что в пуле действительно не менее n потоков.

- При реализации на языках JVM можно именовать классы и интерфейсы сообразно нотации выбранного языка, а также реализовывать коллаборативное завершение работы сообразно стеку технологий по согласованию с преподавателем.

3 блок

4. Деканат решил облегчить себе жизнь и заказал матмеху разработку системы, в которую преподавателями и студентами заносится информация о зачётах у последних. Вам поручено реализовать ядро этой системы, удовлетворяющей следующим критериям:

- Зачёты не дифференцированы, либо они есть, либо их нет.
- Зачёт однозначно идентифицируется парой (идентификатор_студента, идентификатор_курса). Оба идентификатора – длинные целые (64 бита) без дополнительных ограничений.
- Общее число пользователей системы – несколько тысяч.

Система должна поддерживать одновременную и непротиворечивую работу с ней нескольких пользователей.

```
public interface IExamSystem
{
 public void Add(long studentId, long courseId);  public
void Remove(long studentId, long courseId);  public bool
Contains(long studentId, long courseId);  public int
Count { get; }
}
```

Предложить две различные реализации указанного интерфейса с различными подходами к организации взаимодействия между потоками. Использование библиотечных коллекций для организации конкурентного доступа не допускается. Не допускаются реализации с помощью всеобъемлющих высокоуровневых способов вроде за исключением, возможно, Count:

```
public void Add(long studentId, long courseId)
{
 lock(this)
 {
  …
 }
}
```

Обернуть результат в Web API (например, с помощью соответствующей технологии ASP.NET или аналогичной) и Docker-образ.

Провести нагрузочное тестирование получившегося образа с помощью подходящих технологий исходя из следующего распределения запросов: 90% всех вызовов – Contains, 9% - Add, 1% - Remove. Каждый клиент посылает запросы один за другим сразу после окончания предыдущего. Оформить результаты отдельным файлом:

• Указать конфигурацию запуска образа.

• Построить коробчатые диаграммы (она же boxplot) распределения времени выполнения запросов каждого вида в отсутствие другой нагрузки и при двух заданных уровнях нагрузки, исчисляемых в общем количестве запросов к серверу в секунду.

• Найти число клиентов, приводящее к отказу от обслуживания по некоторому таймауту (например, 10, 30 или 60 секунд), указать хотя бы примерное количество записей в словаре в этот момент.

4 блок

5. Написать приложение для peer-to-peer чата. Клиенты устанавливают связь друг с другом напрямую с помощью сокетов. Подключение третьего клиента к одному из двух других уже подключенных клиентов приводит к тому, что три клиента объединяются в единое информационное пространство, и сообщение от одного клиента видно всем остальным. Число подключающихся таким образом клиентов не ограничено. Предусмотреть в полном объёме сопутствующую обработку ошибок. Для сетевого взаимодействия использовать класс Socket (т.е. использование классов TcpClient, TcpListener и UdpClient запрещено). Реализовать графический интерфейс. Предусмотреть возможность выхода из программы и освобождение ресурсов.