Задачи 4 семестра

1 блок

1. Файберы – это потоки внутри потока, которые надо переключать вручную (https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms682661%28v=vs.85%29.aspx). Пример их реализации доступен в приложении 1. ProcessManagerFramework.cs содержит модель процесса операционной системы, где периоды работы сменяются периодами операций ввода-вывода, и при этом априори невозможно узнать, какова будет длительность этих интервалов.

В операционных системах Windows 3.1 и ранних MacOS не было понятия вытесняющей многозадачности. Вместо этого управление отдавалось другому процессу добровольно и вручную – например, когда процесс находился в ожидании операции ввода-вывода. Это реализовано в модели процесса.

Требуется:

- Создать несколько экземпляров класса Process;
- Сделать две реализации метода ProcessManager.Switch – с приоритетным и бесприоритетным алгоритмом диспетчеризации;
- По завершению всех процессов удалить все файберы кроме основного и корректно выйти из программы.

2. 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

3. 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).

4. Реализовать объект ThreadPool, реализующий паттерн «пул потоков». Число потоков задаётся константой в классе пула. Добавление задачи осуществляется с помощью нестатического метода класса пула Enqueue(Action a). Класс должен быть унаследован от интерфейса IDisposable и корректно освобождать ресурсы при вызове метода Dispose().

3 блок

5. 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

6. 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.