0: Добавить свою папку в GIT.

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 проекте Fibers. 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 с применением MPI.NET:

a) чёт-нечётная сортировка;
b) быстрая сортировка;
c) быстрая сортировка с использованием регулярного набора образцов;
d) алгоритм Флойда;
e) 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 $V^2/2$. 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.

3: Реализовать решение упрощённой задачи «производитель-потребитель» (буфер не имеет верхней границы) с указанным средством синхронизации:

- 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().

5: С помощью стандартной или своей версии концепции Future сделать две различных по схеме распараллеливания реализации интерфейса IVectorLengthComputer для подсчёта модуля вектора, представленного массивом целых:

```
public interface IVectorLengthComputer
{
    int ComputeLength(int[] a);
}
```

В отдельном файле приложить оценки ускорения и эффективности каждой схемы при работе на N процессорах. Для набора формул можно пользоваться Microsoft Equation 3.0, MathType 5, TeX или иные средства.

6: 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);
}
```

Предложить две различные реализации указанного интерфейса с различными подходами к организации взаимодействия между потоками. Сравнить их быстродействие из соотношения, что 90% всех вызовов – Contains, 9% - Add, 1% - Remove. Использование библиотечных коллекций для организации конкурентного доступа не допускается. Не допускаются реализации с помощью всеобъемлющих высокоуровневых способов вроде:

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

7: Написать клиент-серверное приложение с клиентской частью на WPF, WinForms или HTML.

Клиентская часть:

- Получает от сервера список известных ему фильтров;
- Позволяет загружать изображение из файла и отображать его на форме;
- Позволяет отправлять изображение на сервер для применения к нему фильтра;
- Во время ожидания ответа от сервера интерфейс остаётся отзывчивым (то есть, например, можно нажимать на кнопки);
- Во время ожидания ответа от сервера есть возможность отменить предыдущее задание;
- Во время ожидания ответа от сервера отображает текущую степень готовности на контроле вроде ProgressBar.

Серверная часть:

- При запуске загружает список доступных фильтров из конфигурационного файла;
- Выдаёт список доступных фильтров на запрос клиентского приложения;
- Обрабатывает получаемые от клиентских приложений изображения выбранным фильтром и возвращает результат;
- Каждую секунду возвращает клиентскому приложению процент готовности текущего задания.

Провести нагрузочное тестирование серверного приложения, выложить приложение или скрипт для тестирования в Git и оформить результаты отдельным файлом:

- Построить график распределения времени выполнения запросов при фиксированном размере изображения в отсутствие другой нагрузки и при двух заданных уровнях нагрузки, исчисляемых в запросах в секунду.
- Построить графики среднего и медианного времени выполнения запросов при различном общем числе пикселов в изображении в отсутствие другой нагрузки и при двух заданных уровнях нагрузки, исчисляемых в запросах в секунду.
- Найти число клиентов, приводящее к отказу от обслуживания, для некоторого фиксированного размера изображения.

Работы с закоммиченными папками \bin, \obj и файлами с расширением .suo не принимаются.