*Курсивом отмечены задачи, формулировка и содержание которых может существенно поменяться*

0: Добавить свою папку в GIT.

1: Реализовать указанный параллельный алгоритм с применением MPI.NET:

1. чёт-нечётная сортировка;
2. быстрая сортировка;
3. быстрая сортировка с использованием регулярного набора образцов;
4. алгоритм Флойда;
5. 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: 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.

3: Реализовать решение упрощённой задачи «производитель-потребитель» с указанным средством синхронизации:

* Производители – объекты, кладущие некоторые объекты (например, числа, строки или более сложные объекты-заявки) в экземпляр класса List<T>;
* Потребители – объекты, извлекающие заявки из экземпляра List<T>;
* Между двумя добавлениями и двумя изъятиями вставляется пауза;
* Количество производителей и потребителей задаётся константами;
* При запуске программы создаются производители и потребители. Они прекращаюют 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. Добавление задачи осуществляется с помощью нестатического метода Enqueue(Action a). Класс должен быть унаследован от интерфейса IDisposable и корректно освобождать ресурсы при вызове метода Dispose().

5: В разработке.

6: С помощью стандартной или своей версии концепции Future сделать две различных по схеме распараллеливания реализации интерфейса IArraySum:

public interface IArraySum

{

int Sum(int[] a); // суммирует все числа в массиве

}

7: Распараллелить реализацию фильтров на C# с использованием следующего алгоритма барьерной синхронизации:

а) tournament tree barrier с индексной арифметикой для навигации по дереву;

b) dissemination barrier

8: 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.

Код задачи 7 можно использовать для обработки изображений.

9: Провести нагрузочное тестирование серверного приложения из задачи 8:

* Построить график распределения времени выполнения запросов при фиксированном размере изображения.
* Построить графики среднего и медианного времени выполнения запросов при различном общем числе пикселов в изображении.
* Найти число клиентов, приводящее к отказу от обслуживания, для некоторого размера изображения.

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