**Лабораторная номер 5.** Многопоточность в .NET

**Цель:** Изучение работы с потоками в среде .NET

**Язык: С#** *(или любой другой язык платформы, но это на любителя)***.**

**Теория:**

Типы, необходимое для работы с потоками, находится в пространстве имен System.Threading.

Основные классы: Поток(Thread), Пул потоков (ThreadingPool), Задача (Task - .NET 4), объекты синхронизации (Mutex, ReadWriteLock, AutoResetEvent, ManualResetEvent, Semaphore и др), фоновая операция (System.ComponentModel.BackgroundWorker).

TPL – библиотека типов, полезных для построения многопоточных программ, например, параллельных циклов, потоков данных, параллельный LINQ и пр.

**Основные свойства и методы потоков:**IsAlive – является ли поток работающим в настоящее время (или уже завершился)

IsBackground – является ли поток фоновым, иначе основным. Фоновые потоки не препятствуют закрытию приложения, если все основные потоки завершены – приложение завершается. По умолчанию создаваемые потоки являются основными.

IsThreadPoolThread – принадлежит ли поток пулу.

Name – позволяет получить или установить имя (полезно для отладки)

Start() – запускает поток на выполнение. Метод, исполняемый потоком, задается в его конструкторе.

Join() – ожидает завершения работы потока (приведет к блокировке UI до завершения задачи, если вызвать в UI потоке!)

**UI поток**

В GUI приложениях всегда есть один поток, отвечающий за работу с GUI. Работа с элементами интерфейса из других потоков, как правило, недопустима, и должна выполняться специальными методами. (В том числе, например, нельзя изменить текст на TextBox из потока, отличного от UI).   
см., например, stackoverflow.com/questions/661561/how-to-update-the-gui-from-another-thread-in-c

**Пул потоков**

Так как создание потока является тяжелым процессом, можно использовать пул (ThreadPool) для быстрого доступа к потокам. Все потоки в пуле являются фоновыми. По умолчанию в пуле создается некоторое количество потоков, при их нехватке автоматически будет создано нужное количество дополнительных потоков. Доступное в настоящее время количество потоков можно узнать методом GetAvailableThreads();

Метод QueueUserWorkItem() позволяет поставить задачу (делегат типа WaitCallback) для обработки в очередь пула. Обработка начнется после того, как появится доступный поток в пуле.

Так же можно использовать следующие средства для запуска потоков из пула:

* Методы BeginInvoke() и EndInvoke() для запуска произвольного делегата асинхронно.
* Класс BackgroundWorker
* Библиотеку TPL (Task Parallel Library)

Кроме того, пулом активно пользуются некоторые компоненты .NET: WCF,ASP, PLINQ и другие.

**Примитивы и методы синхронизации.**

Структура lock(object) (Monitor.Enter/Monitor.Exit) - основной примитив .NET, позволяющий ограничить единовременный доступ к участку кода (критическая секция). Один из самых быстрых примитивов.

Mutex – позволяет синхронизировать потоки различных процессов

Semaphore, ReadWriteLock, ..

**Сигналы (события)**

AutoResetEvent, ManualResetEvent – аналоги WinAPI Event (медленные, межпроцессные)

ManualResetEventSlim (.NET 4) – быстрый (внутрипроцессный, через WaitAndPulse)

CountdownEvent (.NET 4) – срабатывает когда наберется необходимое количество сигналов.

Barries (.NET 4)– барьер выполнения потоков (все потоки будут синхронизированы на барьере)

Wait and Pulse – пара методов, позволяющих реализовать быстрые методы синхронизации с необходимыми условиями.

**Конкуррентные (потокобезопасные) коллекции**

Лежат в пространстве *System.Collections.Concurrent.*

Самые распространенные:

* ConcurrentDictionary<TKey, TValue>
* ConcurrentQueue
* ConcurrentStack
* ConcurrentBag

**Виды асинхронности в .NET:**Вместо ручного создания потоков и управления ими в .NET рекомендуется использовать технологий асинхронного программирования, коих в .NET уже несколько поколений.

* APM (*Asynchronous Programming Model*) – основан на асинхронных делегатах. Исторически первый метод, усложняет код. Предполагает создания семейства методов BeginOperation/EndOperation . *Не рекомендую.*
* EAP (*Event-based Asynchronous Pattern*) - появился в .NET 2.0. Предполагает создание методов с суффиксом Async (OperationAsync), возвращающих специальный объект, реализующий асинхронный интерфейс IAsyncResult. *Пользуйтесь, если у вас старая версия платформы или Mono.*
* TAP (*Task-based Asynchronous Model*) (.NET 4) – новая модель, основанная на задачах (Task). Задача описывает работу, в отличие от потока, который описывает средство выполнения работы. Для ее поддержки введены новые ключевые слова async (объявление асинхронного метода) и await (начало асинхронной части метода). Метод, помеченный async, выполняется синхронно, пока не встретит await, оставшаяся часть метода будет выполнена потоком из пула. Дождаться ее результата можно вызвав Wait на объекте, возвращаемом async методом. *А вот эта технология – ваш друг, или пытается выглядеть им.*
* Некоторые библиотеки реализуют собственные варианты асинхронности, аналогичные перечисленным по функциональности, но отличающиеся по интерфейсу, и, таким образом, не совместимые с ними. *Не делайте так.*

**Лабораторная работа 6.**

Работа с «тяжелыми» задачами, взаимодействие с UI потоком

**Задания на лабораторную работу:**

**1. Использовать пул потоков для выполнения набора типовых задач**

Реализовать UI, отображающее ход выполнения и статус некоторого количества «тяжелых» задач и общее управление ими (одна кнопка старт/стоп на все)

В качестве «тяжелых» задач можно взять какую либо файловую операцию (копирование, архивация файла, поиск файла на диске, поиск текста в файле, вычисление контрольной суммы файла), работу по сети (скачивание файлов, рекурсивное скачивание сайтов) или вычисление (проверка простоты, факторизация чисел), которые требуют нескольких секунд на выполнение.

В интерфейсе приложения должен быть список задач, напротив которой будет отображаться статус (ждет, обрабатывается, выполнена), либо 3 отдельных списка задач.  
После нажатия кнопки «старт» система должна брать потоки из пула для выполнения задач и выполнять их по очереди, отображая статус. Количество одновременно обрабатываемых задач должно быть ограничено (можно вынести настройку в интерфейс).

**2. Реализовать индивидуальное управление набором задач**

Реализовать приложение, позволяющее через UI запускать/приостанавливать/останавливать несколько фоновых задач (каждую задачу индивидуально), с отображением их статуса

Должны быть реализованы: список задач, их статус (ждет, работает, приостановлена, ошибка) , кнопки «пауза», «возобновить», «отменить» (доступны, если выделена задача в списке), «запустить» (запускает новую операцию).

**3. Работа по сети, реализация клиента и сервера для обмена данными**

Реализовать клиент-серверное приложение, использующее сокеты для передачи данных. Работа с сокетами должна вестись в отдельном потоке, на главной форме должен отображаться статус процесса (количество отправленных/полученных байт), приложение должно корректно обрабатывать потерю соединения, выводить соответствующее сообщение и завершать поток обмена данными.

4. Реализовать передачу данных в дуплексном режиме, отображение прогресса отправки и получения данных

5. Реализовать сценарий обработки данных, требующий синхронизации нескольких потоков

(тут можно делать консольное приложение)

Например, один поток последовательно считывает строки файла, затем, группа потоков параллельно обрабатывает их (считает md5, делает замену текста, вычисляет расстояние ливенштейна до заданной строки или что то еще, главное, что выполняемая функция должна быть относительно медленной), после чего результаты вычисления записываются в выходной файл в том же порядке.   
Продемонстрировать, выполнение этой задачи в одном потоке происходит дольше.

**6** ИспользованиеBackgroundWorker (для самых маленьких)**:**

Используя BackgroundWorker реализовать приложение, выполняющее длительную задачу и отображающую прогресс ее выполнения (Progress-bar) в основном окне.

Параметры задачи должны задаваться в основном потоке приложения до запуска операции.  
Необходимо реализовать возможность приостановки задачи (пауза) и ее отмены, обработку исключений в задаче (имитировать исключение в процессе работы).

**7.** Реализовать алгоритм из 1-4 лабораторной, но на .NET, сравнить скорость работы с С++  
(тут нужно делать консольное приложение)

**Баллы:**

Работающая программа: 4б - 6б

Полезная работающая программа: по обстоятельствам

Рассказать, как работает: 2б

Эксперименты с реализацией (сравнить скорость и/или продемонстрировать реализации с разными видами асинхронности). 2б