**Лабораторная работа №22**

**Организация многопоточной обработки на основе класса Thread**

**Теоретические сведения**

C# поддерживает параллельное выполнение кода через многопоточность. Поток – это независимый путь исполнения, способный выполняться одновременно с другими потоками.

Программа на C# запускается как единственный поток, автоматически создаваемый CLR и операционной системой (“главный” поток), и становится многопоточной при помощи создания дополнительных потоков.

CLR назначает каждому потоку свой стек, так что локальные переменные хранятся раздельно. Вместе с тем потоки разделяют данные, относящиеся к тому же экземпляру объекта, что и сами потоки. Иногда необходимо выполнить блокировку участка кода функции на время обработки данных. Выполнение такой операции состоит в получении эксклюзивной блокировки на время чтения и записи разделяемых полей. C# обеспечивает это при помощи оператора lock.

Когда два потока одновременно борются за блокировку (в нашем случае объекта locker), один поток переходит к ожиданию (блокируется), пока блокировка не освобождается. В данном случае это гарантирует, что только один поток может одновременно исполнять критическую секцию кода, и "Done" будет напечатано только один раз. Код, защищенный таким образом от неопределѐнности в плане многопоточного исполнения, называется потокобезопасным.

Временная приостановка (блокирование) – основной способ координации, или синхронизации действий потоков. Ожидание эксклюзивной блокировки – это одна из причин, по которым поток может блокироваться. Другая причина – если поток приостанавливается (Sleep) на заданный промежуток времени:

Thread.Sleep(TimeSpan.FromSeconds(30)); // Блокировка на 30сек

Также поток может ожидать завершения другого потока, вызывая его метод Join:

|  |
| --- |
| Thread t = new Thread(Go); // Go – статический метод t.Start();  t.Join(); // Ожидаем завершения потока |

Будучи блокированным, поток не потребляет ресурсов CPU.

Управление многопоточностью осуществляет планировщик потоков, эту функцию CLR обычно делегирует операционной системе. Планировщик потоков гарантирует, что активным потокам выделяется соответствующее время на выполнение, а потоки, ожидающие или блокированные, к примеру, на ожидании эксклюзивной блокировки, или пользовательского ввода – не потребляют времени CPU.

На однопроцессорных компьютерах планировщик потоков использует квантование времени – быстрое переключение между выполнением каждого из активных потоков. Это приводит к непредсказуемому поведению, как в самом первом примере, где каждая последовательность символов „X‟ и „Y‟ соответствует кванту времени, выделенному потоку. В Windows XP типичное значение кванта времени – десятки миллисекунд – выбрано как намного большее, чем затраты CPU на переключение контекста между потоками (несколько микросекунд).

Говорят, что поток вытесняется, когда его выполнение приостанавливается из-за внешних факторов типа квантования времени. В большинстве случаев поток не может контролировать, когда и где он будет вытеснен. Потоки vs. процессы

Все потоки одного приложения логически содержатся в пределах процесса – модуля операционной системы, в котором исполняется приложение.

В некоторых аспектах потоки и процессы схожи – например, время разделяется между процессами, исполняющимися на одном компьютере, так же, как между потоками одного C#-приложения. Ключевое различие состоит в том, что процессы полностью изолированы друг от друга. Потоки разделяют память (кучу) с другими потоками этого же приложения. Благодаря этому один поток может поставлять данные в фоновом режиме, а другой – показывать эти данные по мере их поступления.

**Создание и запуск потоков**

Для создания потоков используется конструктор класса Thread, принимающий в качестве параметра делегат типа ThreadStart, указывающий метод, который нужно выполнить. Делегат ThreadStart определяется так:

public delegate void ThreadStart();

Вызов метода Start начинает выполнение потока. Поток продолжается до выхода из исполняемого метода. Вот пример, использующий полный синтаксис C# для создания делегата ThreadStart:

class ThreadTest

{

static void Main()

{

Thread t = new Thread(new ThreadStart(Go));

t.Start(); // Выполнить Go() в новом потоке.

Go(); // Одновременно запустить Go() в главном потоке.

} static void Go() { Console.WriteLine("hello!"); }

Поток можно создать, используя для присваивания значений делегатам более удобный сокращенный синтаксис C#:

|  |
| --- |
| static void Main()  {  Thread t = new Thread(Go); // Без явного использования ThreadStart t.Start();  ...  }  static void Go() { ... } |

Поток имеет свойство IsAlive, возвращающее true после вызова Start() и до завершения потока.

Поток, который закончил исполнение, не может быть начат заново.

**Именование потоков**

Поток можно поименовать, используя свойство Name. Это предоставляет большое удобство при отладке: имена потоков можно вывести в Console.WriteLine и увидеть в окне Debug – Threads в Microsoft Visual Studio. Имя потоку может быть назначено в любой момент, но только один раз – при попытке изменить его будет сгенерировано исключение.

Главному потоку приложения также можно назначить имя – в следующем примере доступ к главному потоку осуществляется через статическое свойство CurrentThread класса Thread:

|  |
| --- |
| class ThreadNaming  {  static void Main()  {  Thread.CurrentThread.Name = "main"; Thread worker = new Thread(Go); worker.Name = "worker"; worker.Start();  Go();  }  static void Go()  {  Console.WriteLine("Hello from " + Thread.CurrentThread.Name);  }  } |

По умолчанию потоки создаются как основные, что означает, что приложение не будет завершено, пока один из таких потоков будет исполняться. C# также поддерживает фоновые потоки, они не продлевают жизнь приложению, а завершаются сразу же, как только все основные потоки будут завершены.

Статус потока переключается с основного на фоновый при помощи свойства IsBackground

**Приоритеты потоков**

Свойство Priority определяет, сколько времени на исполнение будет выделено потоку относительно других потоков того же процесса. Существует 5 градаций приоритета потока:

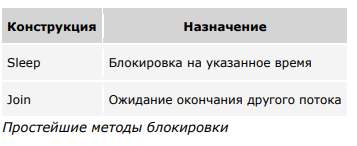
enum ThreadPriority { Lowest, BelowNormal, Normal, AboveNormal, Highest }

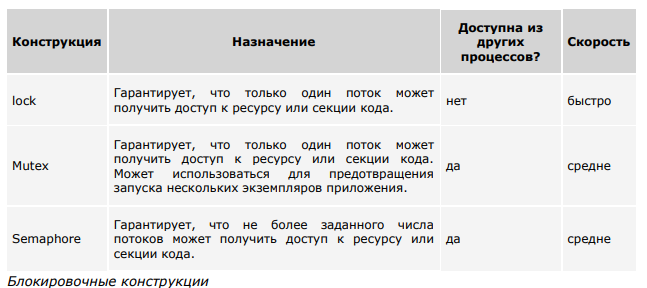
Значение приоритета становится существенным, когда одновременно исполняются несколько потоков.

Установка приоритета потока на максимум еще не означает работу в реальном времени

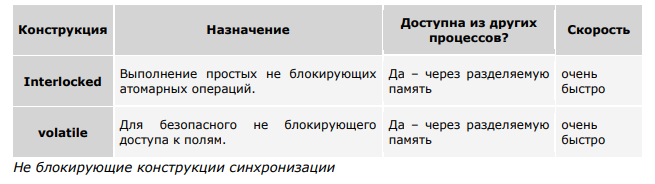
(real-time), так как существуют еще приоритет процесса приложения.

Важнейшие средства синхронизации В следующих таблицах приведена информация об инструментах .NET для координации (синхронизации) потоков:



****

****

****

**Блокировка**

Когда поток остановлен в результате использования конструкций, перечисленных в вышеприведенных таблицах, говорят, что он блокирован. Будучи блокированным, поток немедленно перестает получать время CPU, устанавливает свойство ThreadState в WaitSleepJoin и остается в таком состоянии, пока не разблокируется. Разблокировка может произойти в следующих четырех случаях:

* выполнится условие разблокировки;
* истечет таймаут операции (если он был задан);
* по прерыванию через Thread.Interrupt;  по аварийному завершению через Thread.Abort.

Поток не считается блокированным, если его выполнение приостановлено нерекомендуемым методом Suspend.

**Sleeping**

Вызов Thread.Sleep блокирует текущий поток на указанное время (либо до прерывания):

static void Main()

{

Thread.Sleep(0); // отказаться от одного кванта времени CPU

Thread.Sleep(1000); // заснуть на 1000 миллисекунд

Thread.Sleep(TimeSpan.FromHours(1)); // заснуть на 1 час

Thread.Sleep(Timeout.Infinite); // заснуть до прерывания }

Если быть более точным, Thread.Sleep отпускает CPU и сообщает, что потоку не должно выделяться время в указанный период. Thread.Sleep(0) отпускает CPU для выделения одного кванта времени следующему потоку в очереди на исполнение.

**Задание на лабораторную работу**

**Вариант 1**

Поиск указанной строки в указанном файле. Обработка одной строки в порожденном потоке.

**Вариант 2**

Умножение матрицы на вектор. Обработка одной строки матрицы - в порожденном потоке.

**Вариант 3**

Поиск всех простых чисел (простым называется число, которое является своим наибольшим делителем) в указанном интервале чисел, разделенном на несколько диапазонов. Обработка каждого диапазона производится в порожденном потоке.

**Вариант 4**

Напишите программу, которая содержит три метода: метод Main, метод для подсчета факториала числа (число N задается с клавиатуры) и метод для подсчета суммы целых чисел от 1 до N включительно. В методе Main создайте и запустите два потока: первый для выполнения метода, рассчитывающего факториал, второй – для выполнения метода, подсчитывающего сумму.

**Вариант 5**

Вычислить произведение матриц А (m x n) и В (n x k). Элементы матрицы произведения (С= А х В) вычисляются параллельно однотипными потоками

**Вариант 6**

Вычислить сумму матриц А (n x n) и В (n x n). Элементы матрицы cуммы (С= А + В) вычисляются параллельно однотипными потоками.

**Вариант 7**

Контрольная сумма. Для нескольких файлов (разного размера) требуется вычислить контрольную сумму (сумму кодов всех символов файла). Обработка каждого файла выполняется в отдельном потоке.

**Вариант 8**

Вычислить 10-ю степень двойки 1 - сложением, умножением и просто возведением в степень. Каждый способ вычисления оформить в виде потока.

**Вариант 9**

Создать два потока. Первый ищет числа Фибоначчи (каждое последующее число равно сумме двух предыдущих чисел), второй простые числа. Результаты работы каждого потока сохраняются в отдельный файл. После остановки потока – программа производит анализ файлов, выводит их на экран, а также показывает количество найденных чисел Фибоначчи и простых чисел.

**Вариант 10**

Создать приложение, выполняющее сортировку массива данных. Массив разбить на несколько частей. Сортировку каждой части массива производить в отдельном порожденном потоке.

**Вариант 11**

Создать приложение, выполняющее сортировку массива данных и визуального отображения процесса сортировки на экране. Первый поток производит сортировку по возрастанию, второй по убыванию.

**Вариант 12**

Создать два потока. Первый поток производит запись в файл случайных данных. Второй производит чтение данных из этого файла и вывод их на экран.