МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

по дисциплине
«Введение в технологии высокопроизводительных вычислений»

для студентов направления
09.03.02 «Информационные системы и технологии»

Ставрополь 2025

Оглавление

ВВЕДЕНИЕ

Развитие информационных технологий порождает новые аппаратные и программные архитектуры информационных систем. Увеличени производительности серверов не ограничивается ростом тактовой частоты процессоров – это процесс многокомпонентный: основную роль в увеличении производительности информационных систем играют новые концептуальные схемы взаимодействия вычислителей. При этом учитывается характер вычислителя: узел кластера, процессор, ядро, абстрактный класс задачи или потока. Каждое направление порождает целые семейства технологий построения высокопроизводительных вычислительных систем.

Разработчику информационных систем доступны следующие технологии для построения параллельных и распределенных вычислительных систем:

– вычислительные системы с общей памятью; соответственно программные системы ориентированные на работу в данной модели;

– вычислительные системы, ориентированные на работу в модели распределенной памятью и програмнные компоненты данной архитектуры;

– кластерные технологии;

– высокопроизводительные системы, программным ядром которых являются базы данных;

– вычислительные системы, основанные на графических процессорах;

– программные комплексы функционирующие в многозадачной и многопоточной среде.

Данные направления не исчерпывают всех современных подходов построения высокопроизводительных систем, но образуют группы технологических решений, рассмотрение которых и работа с которыми позволяет изучить и получить практический опыт в сфере рпзработки и построения высокопроизводительных истем.

ЛАБОРАТОРНАЯ РАБОТА 1. ПРИМЕНЕНИЕ ДЕЛЕГАТОВ

1. Цель и содержание

Цель лабораторной работы: изучить возможности применения делегатов в языке C#.

Задачи лабораторной работы:

– освоить принципы работы с делегатами;

– освоить основные направления применения делегатов;

– изучить способы использования делегатов совместно с потоками.

2. Теоретическое обоснование

2.1 Назначение типов делегатов

Делегаты предназначены для ситуаций, когда нужно передать метод другому методу. Бывают случаи, когда методу потребуется обращение к другому методу, но определить к какому именно на этапе компиляции не представляется возможным. Эта информация доступна только во время выполнения, а потому должна быть передана первому методу в виде параметра.

Подобное поведение свойственно в следующих случаях:

1. Запуск потоков и задач. В C# существует возможность сообщить компилятору, что нужно запустить некоторую новую последовательность исполнения параллельно той, что работает в данный момент. Такая последовательность называется потоком (thread), а его запуск осуществляется с помощью метода Start одного из базовых классов – System.Threading.Thread. Если компьютеру сообщается о необходимости запуска новой последовательности исполнения, то также нужно сообщить, откуда именно ее следует начать, т.е. вызов какого метода должен ее

запускать. Другими словами, метод Thread.Start ( ) должен получить параметр, который указывает метод, подлежащий вызову потоком.

2. Обобщенные библиотечные классы. Многие библиотеки содержат код для выполнения разнообразных стандартных задач. Обычно такие библиотеки могут быть самодостаточными – в том смысле, что при их написании вы точно знаете, как следует решать задачи. Однако иногда задача может включать подзадачу, о которой знает только индивидуальный клиентский код, использующий эту библиотеку. Например, требуется написать класс, принимающий массив объектов и сортирующий их по возрастанию. Частью процесса сортировки должно быть повторяющееся сравнение двух объектов из массива, чтобы определить, какой из них нужно расположить первым. Если необходимо обеспечить классу возможность сравнивать любые объекты, то он не может знать заранее, как выполнять такие сравнения. Только клиентский код, использующий класс, может сообщить ему, как следует выполнять сравнение конкретных объектов, массив которых необходимо отсортировать. Клиентский код должен передать вашему классу подробности относительно того, какой метод вызвать для выполнения сравнения.

3. События. Главная идея здесь состоит в том, что часто приходится иметь дело с кодом, который должен быть проинформирован о возникновении каких-то событий. В программировании графического интерфейса пользователя полно подобных ситуаций. Когда событие происходит, исполняющая среда должна знать, какой метод необходимо вызвать. Это делается передачей методу, обрабатывающему события, параметра-делегата.

Как и в случае классов для использования делегата, его необходимо заранее объявить. Синтаксис объявления делегатов выглядит следующим образом:

delegate $< тип\_возврата > Имя\_типа\_делегата$ ([*параметры*]) .

Например:

```
delegate void MyFirstDelegate(int pParam);
```

В данном примере определяется делегат, экземпляр которого принимает один параметр типа int и возвращает значение типа void.

Еще несколько примеров объявления делегатов:

```
delegate int Summ (int a, int v);
delegate string ConverterFromFloat (float D);
delegate int[] ToVector(int a, int b, int c);
delegate string GetAString();
```

Следует понимать, что при объявлении делегата с использованием ключевого слова delegate фактически создается специализированный класс. Делегаты создаются как классы, унаследованные от System.MulticastDelegate.

```
delegate string GetAString();

class Program
{
    static void Main(string[] args)
    {
        int x = 40;

        // переменная firstStringMethod
        //инициализируется значением x.ToString(),
        GetAString firstStringMethod = new GetAString(x.ToString);

        // приведенный оператор эквивалентен следующему:
        // Console.WriteLine("Строка равна " + x.ToString ());
        Console.WriteLine("Строка равна " + firstStringMethod());
    }
}
```

Применение скобок к экземпляру делегата это все равно что вызов метода Invoke ( ), то есть следующие сроки выполняют одно и тоже действие:

firstStringMethod ( );

firstStringMethod.Invoke ( );

Чтобы сократить код, в каждом месте, где требуется экземпляр делегата, можно просто передать имя адреса. Это называется выведением делегата (delegate inference). Это средство C# работает до тех пор, пока компилятор способен разрешить экземпляр делегата в специфический тип. В

предыдущем примере переменная firstStringMethod типа GetAsString инициализируется новым экземпляром делегата GetAsString:

GetAString firstStringMethod = new GetAString(x.ToString);

Сделать то же самое можно, просто передав переменной firstStringMethod имя метода в переменной x:

GetAString firstStringMethod = x.ToString;

В обоих случаях компилятор C# создает один и тот же код. Компилятор обнаруживает, что тип делегата требуется firstStringMethod, поэтому создает экземпляр делегата типа GetAString и передает конструктору адрес метода в объекте x.

Очевидно, что дописывать круглые скобки при передаче имени метода недопустимо.

Экземпляр делегата может ссылаться на любой метод — уровня экземпляра или статический – любого объекта любого типа, если сигнатура этого метода совпадает с сигнатурой делегата.

2.2 Делегаты Action<T> и Func<T> библиотеки .NET Framework.

Вместо определения нового типа делегата с каждым типом параметра и возврата можно использовать делегаты Делегаты Action<T> и Func<T>. Обобщенный делегат Делегаты Action<T> предназначен для ссылки на метод, возвращающий мщшв. Этот класс делегата существует в различных вариантах, так что ему можно передавать до 16 разных типов параметров.

Класс Action без обобщенного параметра предназначен для вызова методов без параметров, Action<in T> – для вызова метода с одним параметром, Action<in Tl, in T2> – для вызова метода с двумя параметрами и Action<in Tl, in T2, in T3, in T4, in T5, in T6, in T7, in T8> – для вызова метода с восемью параметрами.

Делегаты Func<T> могут использоваться аналогичным образом. Func<T> позволяет вызывать методы с типом возврата. Подобно Action<T>,

Func<T> определен в разных вариантах для передачи до 16 типов параметров и типа возврата.

Func<out Tresult> – тип делегата для вызова метода с типом возврата, но без параметров, Func<int Tl, out TResult> – для метода с одним параметром, а Func<in Tl, in T2, in T3, in T4, out TResult> – для метода с четырьмя параметрами.

2.3 Лямбда-выражения

Начиная с C# 3.0, доступен новый синтаксис для назначения реализации кода делегатам, называемый лямбда-выражениями (lambda expression). Лямбда-выражения могут использоваться везде, где есть параметр типа делегата. Ниже показан предыдущий пример, в котором применялись анонимные методы, адаптированный для использования лямбда-выражения.

```
static void Main(string[] args)
{
    string mid = ", средняя часть,";
    Func<string, string> lambda = param =>
    {
    param += mid;
    param += " а это добавлено к строке.";
    return param;
    };
    Console.WriteLine(lambda ("Начало строки")) ;
}
```

При использовании лямбда-выражений существует несколько способов определения параметров. Если параметр один, то достаточно его имени. В следующем лямбда-выражении используется параметр по имени s. Поскольку тип делегата определяет параметр string, то s имеет тип string. Реализация вызывает метод String.Format () для возврата строки, которая в конечном итоге выводится на консоль при вызове делегата – изменение регистра текстовой строки «TEST»:

```
static void Main(string[] args)
{
    Func<string, string> oneParam = s
        => String.Format("изменение регистра {0}",
        s.ToUpper());
    Console.WriteLine(oneParam("test"));
}
```

Если делегат использует более одного параметра, имена параметров можно комбинировать внутри скобок. Ниже представлены параметры x и y типа double, определенные делегатом Func<double, double, double>:

```
static void Main(string[] args)
{
    Func<double, double, double> twoParams = (x, y) => x * y;
    Console.WriteLine(twoParams(10, 2));
}
```

Для удобства к именам переменных внутри скобок можно добавлять типы параметров:

```
static void Main(string[] args)
{
    Func<double, double, double> twoParamsWithTypes =
    (double x, double y) => x * y;
    Console.WriteLine(twoParamsWithTypes(100, 2));
}
```

Если лямбда-выражение состоит из единственного оператора, блок метода с фигурными скобками и оператором return не требуются. Компилятор добавляет return неявным образом:

```
Func<double, double> square =
    x => x * x;
```

Можно добавить фигурные скобки, оператор return и точку с запятой. Просто обычно легче читать код без них:

```
Func<double, double> square =
    x =>
    {
        return x * x;
    };
```

Но если нужно включить в реализацию лямбда-выражения несколько операторов, то фигурные скобки и оператор return обязательны:

```
static void Main(string[] args)
{
    string mid = "Начало строки";

    Func<string, string> lambda = param =>
    {
        param += mid;
        param += " а это добавлено к строке.";
        return param;
    };
}
```

3. Методика и порядок выполнения работы

1. Создайте приложение C# в среде MS Visual Studio.

2. В соответствии с вариантом индивидуального задания реализуйте пользовательский тип делегата требуемой сигнатуры и выполните с его использованием вызов нескольких методов (с корректной сигнатурой).

Индивидуальное задание.

Вариант	Делегат (сигнатура)
1	Action<Func<float>, bool, List<float>>
2	Func<Action<char>, bool, double, double>
3	Action<Func<double>, double, double>
4	Func<Action<float>, int, float, bool>
5	Action<Func<bool, int>, char, string>
6	Func<Action<int>, bool, char, string>
7	Action<Func<bool>, double, double>
8	Func<Action<T>, float, float> : T object
9	Action<Func<int>, char, char>
10	Func<Action<int, int>, bool>
11	Action<Func<int>, char>
12	Func<Action<bool>, float, float>
13	Action<Func<char>, int, bool>
14	Func<Action<List<int>>, bool>
15	Action<Func<int>, List<float>>
16	Func<Action<string>, int, int>
17	Action<Func<int, int>, List<string>>
18	Func<Action<string, int>, int, int>
19	Action<Func<bool>, string, int>

20	Func<Action<int>, int, int>
21	Action<Func<double, double, double>, double>
22	Func<Action<int, int>, string>
23	Action<Func<char, int>, List<string>>
24	Func<Action<IEnumerable<T>>, int> : T object
25	Action<Func<int, int, int>, string>

4. Вопросы для защиты работы

1. Что такое тип делегата? Какой аналог типа делегата существует в C++?

2. Опишите основные направления использования делегатов.

3. Какие механизмы технологии Windows Forms реализованы с использованием делегатов?

4. Для чего предназначен тип Action<T>? Чем он отличается от Func<T>?

5. Чем пользовательские делегаты отличаются от библиотечных?

ЛАБОРАТОРНАЯ РАБОТА 2. ПРИМЕНЕНИЕ АСИНХРОННЫХ ДЕЛЕГАТОВ ДЛЯ РЕАЛИЗАЦИИ МНОГОПОТОЧНОСТИ

1. Цель и содержание

Цель лабораторной работы: научиться использовать делегаты для организации многопоточного приложения.

Задачи лабораторной работы:

– научиться объявлять, инициализировать и запускать потоки с использованием пользовательских делегатов;

– научиться запускать потоки с использованием библиотечных делегатов Action<T> и Func<T>;

– научиться запускать параллельные потоки с использованием лямбда-выражений.

2. Теоретическое обоснование

Наиболее простым способом для создания потока является определение делегата и его вызов асинхронным образом. Класс Delegate поддерживает и возможность асинхронного вызова методов. Для решения поставленной задачи класс Delegate создает отдельный поток.

Чтобы посмотреть в действии на асинхронные возможности делегатов, сначала нужно создать метод, выполнение которого занимает определенное время. Например, ниже показан метод TakesAWhile (), который благодаря вызову Thread.Sleep( ) выполняется минимум столько времени, сколько в миллисекундах задано во втором аргументе.

```
static int TakesAWhile(int data, int ms)
{
    Console.WriteLine("TakesAWhile запущен");
    Thread.Sleep(ms);
    Console.WriteLine("TakesAWhile завершен");
    return ++data;
}
```

Чтобы обеспечить вызов этого метода из делегата, понадобится определить тип делегата с тем де самым параметром и возвращаемым типом:

```
public delegate int TakesAWhileDelegate(int data, int ms);
```

Теперь можно применять различные приемы для асинхронного вызова данного делегата и возврата результатов.

Одним из таких приемов является опрос и проверка, завершил ли делегат свою работу. Созданный класс delegate предоставляет метод BeginInvoke ( ), в котором могут передаваться входные параметры, определенные вместе с типом делегата. Метод Beginlnvoke ( ) всегда имеет два дополнительных параметра типа AsyncCallback и object, которые будут рассматриваться позже. Сейчас главный интерес представляет возвращаемый тип метода – IAsyncResult. С помощью IAsyncResult можно извлекать информацию о делегате и проверять, завершил ли он свою работу, что и демонстрируется в примере с применением свойства IsCompleted. Цикл while продолжает выполняться в главном потоке программы до тех пор, пока делегат не завершит работу.

```
static void Main(string[] args)
{
    // синхронный вызов метода
    // TakesAWhileA, 3000);

    // асинхронный вызов метода с применением делегата
    TakesAWhileDelegate dl = TakesAWhile;

    IAsyncResult ar = dl.BeginInvoke(1, 3000, null, null);
    while (!ar.IsCompleted)
    {
        // выполнение еще каких-нибудь Сераций в главном потоке
        Console.Write(".");
        Thread.Sleep(50);
    }
    int result = dl.EndInvoke(ar);

    // вывод результата
    Console.WriteLine("result: {0}", result);
}
```

После запуска этого приложения можно увидеть, что главный поток и поток делегата выполняются параллельно, а после завершения работы потока делегата главный поток прекращает проход по циклу. Если главный поток завершает выполнение, не дожидаясь завершения работы делегата, поток делегата останавливается.

В данном разделе методических указаний рассмотрен самый простой метод запуска метода в отдельном потоке – с использованием асинхронного делегата. Рассмотрен также механизм передачи параметров в поток, выполняемый параллельно (асинхронно); возврат результата из асинхронно выполняемого метода. Необходимо обратить внимание на реализацию механизма ожидания завершения работы асинхронного метода.

3. Методика и порядок выполнения работы

1. Создайте приложение на языке C# в MS Visual Studio.

2. В соответствии с индивидуальным вариантом разработайте требуемый тип делегата (пользовательский, библиотечный или лямбда-выражение).

3. Реализуйте асинхронное выполнение метода на основе разработанного делегата с возможностью мониторинга процесса выполнения, передачи параметров в метод и получения результата работы метода.

Индивидуальное задание.

Вариант	Тип делегата	Решаемая задача (результат метода)	Входные параметры
1	пользовательский	Метод возвращает сумму элементов матрицы целых случайных чисел	Два параметра: размер матрицы
2	библиотечный	Метод возвращает разницу максимального и минимального элементов матрицы целых случайных чисел	Два параметра: размер матрицы
3	лямбда-выражение	Метод возвращает логическое значение, указывающее существует ли заданное число в массиве целых случайных чисел	Два параметра: размер массива и искомый элемент
4	пользовательский	Метод возвращает матрицу случайных битовых значений (0 или 1), формируемую случайным образом	Два параметра: размер матрицы
5	библиотечный	Метод возвращает строку максимальной длины на основе переданного массива строк	Один параметр: массив (или список) строк
6	лямбда-выражение	Метод возвращает подмножество элементов массива случайных чисел, которые делятся на 3	Один параметр: размер исходного массива
7	пользовательский	Метод возвращает число – количество элементов присутствующих в обоих массивах случайных чисел.	Два параметра: размеры массивов
8	библиотечный	Метод возвращает среднее арифметическое элементов матрицы случайных чисел.	Два параметра: размер матрицы
9	лямбда-выражение	Метод возвращает результат шифрования строки: каждый исходный символ строки заменяется шифрованным символом, код которого на n больше кода исходного символа.	Два параметра: исходная строка, число сдвига n
10	пользовательский	Метод возвращает статистику для строки: список пар (символ строки, количество вхождений)	Один параметр: анализируемая строка

11	библиотечный	Метод возвращает разницу максимального и минимального элементов матрицы.	Два параметра: размер матрицы
12	лямбда-выражение	Метод возвращает подмножество элементов массива случайных чисел, которые являются четными	Один параметр: размер исходного массива
13	пользовательский	Метод возвращает количество нечетных элементов в матрице случайных чисел	Два параметра: размер матрицы
14	библиотечный	Метод возвращает подмножество элементов массива случайных чисел, которые отличаются от заданного числа не более чем на 4.	Два параметра: размер массива, значеное число
15	лямбда-выражение	Метод возвращает логическое значение, указывающее существует ли заданный символ в строке	Два параметра: строка и искомый символ
16	пользовательский	Метод возвращает два числа – максимальные элементы массивов случайных чисел.	Два параметра: размеры массивов
17	библиотечный	Метод возвращает скадярное произведение двух случайных векторов.	Один параметр: размер векторов
18	лямбда-выражение	Метод возвращает сумму двух матриц случайных чисел	Два параметра: размер матриц
19	пользовательский	Метод возвращает количество вхождений заданного элемента в матрице вещественных чисел	Два параметра: матрица и целевой элемент
20	библиотечный	Метод возвращает подмножество элементов массива случайных чисел, которые делятся на 6	Один параметр: размер исходного массива
21	лямбда-выражение	Метод возвращает результат сложения двух случайных векторов целых чисел	Один параметр: размер векторов
22	пользовательский	Метод возвращает количество вхождений заданного символа в строке	Два параметра: строка и целевой символ
23	библиотечный	Метод возвращает подмножество четных элементов матрицы случайных чисел	Два параметра: размер матрицы
24	лямбда-выражение	Метод возвращает статистику массива случайных целых чисел: список пар (число массива, количество вхождений)	Один параметр: размер массива

25	пользовательский	Метод возвращает логическое значение, указывающее существует ли заданный элемент в матрице чисел double	Два параметра: матрица и искомый элемент

4. Вопросы для защиты работы

1. Поясните назначение типа IAsyncResult.

2. Для чего используется метод Thread.Sleep( )?

3. Поясните механизм возврата значения из метода асинхронного делегата.

4. Как произвести возврат более одного значения из метода?

5. Какая разница существует между библиотечными делегатами, пользовательскими типами делегатов и лямбда-выражениями? Являются ли эти делегаты взаимозаменяемыми при реализации асинхронного вызова методов?

ЛАБОРАТОРНАЯ РАБОТА 3. ОЖИДАНИЕ ЗАВЕРШЕНИЯ АСИНХРОННОГО МЕТОДА С ИСПОЛЬЗОВАНИЕМ ТАЙМ-АУТА

1. Цель и содержание

Цель лабораторной работы: научиться использовать механизм ожидания завершения работы асинхронного метода с использованием типа IAsyncResult и тайм-аута.

Задачи лабораторной работы:

– научиться использовать механизм тайм-аутов;

– научиться выводить информацию о ходе выполнения асинхронного метода;

– научиться отслеживать выполнение асинхронного метода.

2. Теоретическое обоснование

Другой способ ожидания результата от асинхронного делегата предусматривает применение дескриптора ожидания (wait handle), который ассоциируется с IAsyncResult.

Получить доступ к этому дескриптору ожидания можно через свойство AsyncWaitHandle. Это свойство возвращает объект типа WaitHandle, с помощью которого можно организовать ожидание завершения работы потоком делегата. Метод WaitOne ( ) принимает в качестве первого необязательного параметра значение тайм-аута, в котором можно указать максимальный период времени ожидания. В рассматриваемом здесь примере этот период составляет 50 миллисекунд. В случае возникновения тайм-аута метод WaitOne ( ) возвращает значение false, и проход по циклу while продолжается. Если во время ожидания до тайм-аута дело не доходит, осуществляется выход из цикла while с помощью break и получение

результата методом Endlnvoke ( ). В пользовательском интерфейсе результат выглядит примерно так же, как в предыдущем примере, отличается лишь способ реализации ожидания.

```
static void Main(string[] args)
{
    TakesAWhileDelegate dl = TakesAWhile;
    IAsyncResult ar = dl.BeginInvoke(1, 3000, null, null);
    while (true)
    {
        Console.Write (" . ") ;
        if (ar.AsyncWaitHandle.WaitOne(50, false))
        {
            Console.WriteLine("Можно извлечь результат сейчас");
            break;
        }
    }

    int result = dl.EndInvoke(ar);

    // вывод результата
    Console.WriteLine("result: {0}", result);
}
```

3. Методика и порядок выполнения работы

1. Создайте консольное приложение C# в Visual Studio.

2. Реализуйте приложение с использованием тайм-аута (модифицируйте приложение, полученное в ходе выполнения лабораторной работы №2).

3. Реализуйте механизм вывода информации в консоль о ходе решения задачи асинхронным методом.

4. Вопросы для защиты работы

1. Для чего применяется тип IAsyncresult?

2. Как реализовать ожидание завершения выполнения асинхронного метода с использованием тайм-аута?

3. Поясните назначение метода WaitOne( ).

ЛАБОРАТОРНАЯ РАБОТА 4. ИСПОЛЬЗОВАНИЕ ОБРАТНЫХ АСИНХРОННЫХ ВЫЗОВОВ

1. Цель и содержание

Цель лабораторной работы: научиться использовать обратные асинхронные вызовы для организации ожидания завершения выполнения асинхронного метода.

Задачи лабораторной работы:

– изучить возможности применения обратных асинхронных вызовов для параллельных потоков;

– изучить возможности асинхронного обратного вызова при использовании типа делегата;

– изучить возможности асинхронного обратного вызова при использовании лямбда-выражения.

2. Теоретическое обоснование

Еще один способ для ожидания результатов от делегата заключается в применении так называемого асинхронного обратного вызова (asynchronous callback). С помощью третьего параметра в методе BeginInvoke можно передать метод, удовлетворяющий требованиям делегата AsyncCallback. Делегат AsyncCallback требует определять параметр IAsnycResult и возвращаемый тип void. В рассматриваемом здесь примере третьему параметру назначается адрес метода TakesAWhileCompleted, который удовлетворяет требованиям делегата AsyncCallback. В последнем параметре методу Beginlnvoke можно передать объект, к которому будет производиться доступ из метода обратного вызова. Здесь удобно передать экземпляр самого

делегата, так что метод обратного вызова сможет использовать его для получения результата асинхронного метода.

```
static void Main(string[] args)
{
    TakesAWhileDelegate dl = TakesAWhile;
    dl.BeginInvoke(1, 3000, TakesAWhileCompleted, dl) ;
    for (int i = 0; i < 100; i++)
    {
        Console.Write(".");
        Thread.Sleep(50);
    }
}

static void TakesAWhileCompleted(IAsyncResult ar)
{
    if (ar == null) throw new ArgumentNullException("ar");
    TakesAWhileDelegate dl = ar.AsyncState as TakesAWhileDelegate;

    // Неверный тип объекта
    Trace.Assert (dl != null, "Invalid object type");

    int result = dl.EndInvoke(ar);

    // вывод результата
    Console.WriteLine("result: {0}", result);

}
```

По завершении работы делегата TakesAWhileDelegate сразу же вызывается метод TakesAWhileCompleted ( ). Ожидать результатов внутри главного потока нет никакой необходимости. Завершать главный поток перед завершением работы потоков делегатов может быть необязательно, если только остановка работы потоков делегатов при завершении выполнения главного потока не представляет проблемы.

Метод TakesAWhileCompleted ( ) определяется с типами параметров и возврата, которые указаны в делегате AsyncCallback. Последний параметр, передаваемый в Beginlnvoke ( ), может быть прочитан с помощью ar.AsyncState, а результат получен вызовом в TakesAWhileDelegate метода Endlnvoke.

Вместо того чтобы определять отдельный метод и передавать его методу Beginlnvoke ( ), можно воспользоваться лямбда-выражением.

Параметр ar имеет тип IAsyncResult. В такой реализации нет необходимости присваивать значение последнему параметру метода Beginlnvoke ( ), потому что лямбда-выражение может напрямую получать доступ к переменной d1, находящейся за пределами контекста данного метода. Но блок реализации лямбда-выражения все равно должен вызываться из потока делегата, что может и не быть очевидным при определении метода подобным образом:

```
static void Main(string[] args)
{
    TakesAWhileDelegate dl = TakesAWhile;
    dl.BeginInvoke (1, 3000, ar =>
    {
        int result = dl.EndInvoke(ar);
        // вывод результата
        Console.WriteLine("result: {0}", result);
    },
    null);

    for (int i = 0; i < 100; i++)
    {
        Console.Write(".");
        Thread.Sleep(50);
    }
}
```

Модель программирования и приемы, описанные для асинхронных делегатов – опрос, дескрипторы ожидания и асинхронные вызовы = доступны не только для делегатов. Эта же модель программирования, которая называется шаблоном асинхронного вызова (asynchronous pattern), встречается в разнообразных местах .NET Framework. Например, с помощью метода BeginGetResponse ( ) класса HttpWebRequest можно асинхронно отправлять HTTP-запросы, а с помощью метода BeginExecuteReader ( ) класса SqlCommand – запросы к базе данных. Параметры похожи на те, что можно передавать в методе Beginlnvoke ( ) делегата, а механизмы, применяемые для получения результатов, выглядят точно так же.

3. Методика и порядок выполнения работы

1. Создайте консольное приложение C# в MS Visual Studio.

2. Реализуйте решение задачи в соответствии с индивидуальным вариантом с использованием асинхронного вызова метода. При реализации получения результата работы асинхронного метода используйте механизм обратного асинхронного вызова.

3. Обратите внимание, что обратный вызов можно выполнить как с использованием типа делегата, так и с использованием лямбда-выражения. В задании индивидуального варианта указано, какой способ обратного вызова следует использовать.

Индивидуальное задание.

Вариант	Тип делегата	Решаемая задача (результат метода)	Входные параметры	Метод обратного вызова
1	библиотечный	Метод возвращает подмножество элементов массива случайных чисел, которые делятся на 6	Один параметр: размер исходного массива	лямбда-выражение
2	лямбда-выражение	Метод возвращает результат сложения двух случайных векторов целых чисел	Один параметр: размер векторов	делегат
3	пользовательский	Метод возвращает количество вхождений заданного символа в строке	Два параметра: строка и целевой символ	лямбда-выражение
4	библиотечный	Метод возвращает подмножество четных элементов матрицы случайных чисел	Два параметра: размер матрицы	делегат
5	лямбда-выражение	Метод возвращает статистику массива случайных целых чисел: список пар (число массива, количество вхождений)	Один параметр: размер массива	лямбда-выражение
6	пользовательский	Метод возвращает логическое значение, указывающее существует ли заданный элемент в матрице чисел double	Два параметра: матрица и искомый элемент	делегат

7	пользовательский	Метод возвращает сумму элементов матрицы целых случайных чисел	Два параметра: размер матрицы	делегат
8	библиотечный	Метод возвращает разницу максимального и минимального элементов матрицы целых случайных чисел	Два параметра: размер матрицы	лямбда-выражение
9	лямбда-выражение	Метод возвращает логическое значение, указывающее существует ли заданное число в массиве целых случайных чисел	Два параметра: размер массива и искомый элемент	делегат
10	пользовательский	Метод возвращает матрицу случайных битовых значений (0 или 1), формируемую случайным образом	Два параметра: размер матрицы	лямбда-выражение
11	библиотечный	Метод возвращает строку максимальной длины на основе переданного массива строк	Один параметр: массив (или список) строк	делегат
12	лямбда-выражение	Метод возвращает подмножество элементов массива случайных чисел, которые делятся на 3	Один параметр: размер исходного массива	лямбда-выражение
13	пользовательский	Метод возвращает число – количество элементов присутствующих в обоих массивах случайных чисел.	Два параметра: размеры массивов	делегат
14	библиотечный	Метод возвращает среднее арифметическое элементов матрицы случайных чисел.	Два параметра: размер матрицы	лямбда-выражение
15	лямбда-выражение	Метод возвращает результат шифрования строки: каждый исходный символ строки заменяется шифрованным символом, код которого на n больше кода исходного символа.	Два параметра: исходная строка, число сдвига n	делегат
16	пользовательский	Метод возвращает статистику для строки: список пар (символ строки, количество вхождений)	Один параметр: анализируемая строка	лямбда-выражение

17	библиотечный	Метод возвращает разницу максимального и минимального элементов матрицы.	Два параметра: размер матрицы	делегат
18	лямбда-выражение	Метод возвращает подмножество элементов массива случайных чисел, которые являются четными	Один параметр: размер исходного массива	лямбда-выражение
19	пользовательский	Метод возвращает количество нечетных элементов в матрице случайных чисел	Два параметра: размер матрицы	делегат
20	библиотечный	Метод возвращает подмножество элементов массива случайных чисел, которые отличаются от заданного числа не более чем на 4.	Два параметра: размер массива, значеное число	лямбда-выражение
21	лямбда-выражение	Метод возвращает логическое значение, указывающее существует ли заданный символ в строке	Два параметра: строка и искомый символ	делегат
22	пользовательский	Метод возвращает два числа – максимальные элементы массивов случайных чисел.	Два параметра: размеры массивов	лямбда-выражение
23	библиотечный	Метод возвращает скадярное произведение двух случайных векторов.	Один параметр: размер векторов	делегат
24	лямбда-выражение	Метод возвращает сумму двух матриц случайных чисел	Два параметра: размер матриц	лямбда-выражение
25	пользовательский	Метод возвращает количество вхождений заданного элемента в матрице вещественных чисел	Два параметра: матрица и целевой элемент	делегат

4. Вопросы для защиты работы

1. Поясните назначение каждого параметра метода BeginInvoke( ).

2. Почему при использовании типа делегата в качестве метода обратного вызова последний параметр метода BeginInvoke( ) можно не использовать?

3. Опишите области использования асинхронных делегатов. В каких типах проектов .NET Framework они применимы?

ЛАБОРАТОРНАЯ РАБОТА 5. ПРИМЕНЕНИЕ КЛАССА THREAD

1. Цель и содержание

Цель лабораторной работы: научиться использовать базовые возможности класса потоков Thread.

Задачи лабораторной работы:

– научиться создавать потоки Thread;

– научиться использовать массивы потоков;

– научиться осуществлять мониторинг потоков.

2. Теоретическое обоснование

С помощью класса Thread можно не только создавать потоки, но и управлять ими. Ниже приведен код простого примера создания и запуска нового потока.

```
static void Main()
{
    var tl = new Thread(ThreadMain);
    tl.Start();
    Console.WriteLine("Главный поток.");
}

static void ThreadMain()
{
    Console.WriteLine("Выполняется в потоке.");
}
```

В этом коде конструктор класса Thread перегружен для приема делегата типа ThreadStart или ParameterizedThreadStart. Делегат ThreadStart определяет метод с типом возврата void, не принимающий аргументы. После создания объекта Thread поток может запускаться с помощью метода Start ( ).

После запуска приложения на экран выводятся данные из двух потоков:

This is the main thread.

Running in a thread.

Приведенный выше порядок вывода не гарантируется. Дело в том, что выполнение потоков планируется операционной системой, а это значит, что каждый раз очередность потоков может меняться.

Ранее уже упоминалось, что с асинхронным делегатом можно использовать лямбда-выражение. Его также можно применять и с классом Thread, передавая реализацию метода потока конструктору Thread:

```
static void Main()
{
    var tl = new Thread(() =>
        Console.WriteLine("Выполняется в потоке, id: {0}",
        Thread.CurrentThread.ManagedThreadId));

    tl.Start();
    Console.WriteLine("Главный поток, id: {0}",
    Thread.CurrentThread.ManagedThreadId);
}
```

Теперь в выводе программы будет также присутствовать идентификатор потока:

Главный поток, id: 1.

Выполняется в потоке, id: 3.

3. Методика и порядок выполнения работы

1. Создайте консольное приложение и реализуйте пример решения задачи с использованием потоков.

Пример решения задачи:

Реализовать выполнение лабораторной работы для метода поиска максимального элемента в массиве случайных чисел.

Решение примера:

1.1. Реализуем метод поиска максимального элемента в массиве случайных чисел (рис. 5.1):

```
static void CalcMax()
{
    int threadId = Thread.CurrentThread.ManagedThreadId;

    Console.WriteLine("Поток {0}. Создание массива...", threadId);
    int N = 10;
    int[] mas = new int[N];

    Console.WriteLine("Поток {0}. Инициализация элементов массива...", threadId);
    Random rnd = new Random(threadId);
    for(int i = 0; i < mas.Length; i++)
    {
        mas[i] = rnd.Next(1000);
    }

    Console.WriteLine("Поток {0}. Поиск максимального элемента...", threadId);
    int max = mas[0];
    for (int i = 0; i < mas.Length; i++)
    {
        max = (mas[i] > max) ? mas[i] : max;
    }

    Console.WriteLine("Поток {0}. Максимальный элемент = {1}",
        threadId,
        max);
}
```

Рисунок 5.1 – Метод поиска максимального элемнта в массиве случайных чисел.

1.2. Реализуем функцию Main с использованием массива потоков, каждый из которых будет работать со своей копией случайного массива (рис. 5.2).

```
static void Main(string[] args)
{
    Action method = CalcMax;
    Thread[] tmas = new Thread[20]; // 20 потоков
    for (int i = 0; i < tmas.Length; i++ )
    {
        tmas[i] = new Thread(CalcMax);
        tmas[i].Start();
    }

    Console.ReadKey();
}
```

Рисунок 5.2 – Метод Main для создания множества потоков.

1.3. Запустите приложение, Попробуйте изменить значения количества потоков и размер массива. Обратите внимание на вывод информации о ходе выполнения решения задачи.

2. Создайте консольное приложение с использованием MS Visual Studio.

3. Реализуйте метод для запуска в отдельном потоке (в соответствии с индивидуальным вариантом). Создайте делегат для представления метода (если требуется).

4. В основной программе (функция Main( )) реализуйте создание массива потоков (размер определите самостоятельно). Зтем запустите все элементы массива (потоки) на выполнение.

5. Метод, выполняющийся в параллельных потоках должен выводить информацию о ходе своего выполнения в консоль приложения.

Индивидуальное задание.

Вариант	Метод для реализации
1	Метод вычисления среднего арифметического элементов массива.
2	Метод поиска максимального элемента в матрице.
3	Метод преобразования матрицы случайных чисел: каждый элемент заменяется косинусом элемента матрицы.
4	Метод подсчета количества четных элементов в матрице.
5	Метод вычисления скалярного произведения двух случайных векторов.
6	Метод расчета модуля случайного вектора.
7	Метод преобразования матрицы случайных чисел: четные элементы заменяются тангенсом элемента матрицы.
8	Метод подсчета количества чисел, кратных 3, в массиве случайных чисел.
9	Метод вычисления статистики (количество для каждого входящего символа) по строке.
10	Метод вычисления статистики по случайному массиву (количество элементов для каждого значения массива).
11	Метод преобразования матрицы случайных чисел: нечетные элементы заменяются –1.
12	Метод поиска среднего арифметического элементов матрицы.
13	Метод вычисления минимального значения в матрице случайных чисел.
14	Метод расчета суммы элементов массива случайных чисел.
15	Метод преобразования матрицы случайных чисел: элементы кратные 3 заменяются на квадрат соответствующего элемента.
16	Метод вычисления факториала числа.

17	Метод преобразования матрицы случайных чисел: каждый элемент заменяется синусом элемента матрицы.
18	Метод преобразования матрицы случайных чисел: четные элементы заменяются суммой косинуса и синуса соответствующего элемента матрицы.
19	Метод преобразования матрицы случайных чисел: нечетные элементы заменяются суммой тангенса и котангенса соответствующего элемента матрицы.
20	Метод вычисления статистики по случайной матрице (количество элементов для каждого значения матрицы).
21	Метод подсчета количества четных и нечетных элементов матрицы.
22	Метод вычисления статистики по трехмерному массиву (количество элементов для каждого значения массива).
23	Метод преобразования матрицы случайных чисел: все элементы заменяются суммой косинуса и синуса соответствующего элемента матрицы.
24	Метод вычисления среднего арифметического трехмерного массива случайных чисел.
25	Метод вычисления суммы двух матриц.

4. Вопросы для защиты работы

1. В каком пространстве имен определен класс Thread? Поясните назначение класса Thread.

2. Как получить идентификатор текущего потока?

3. Какой метод осуществляет запуск метода на выполнение в потоке?

4. Для чего необходимы делегаты при использовании класса Thread?

5. Как организовать несколько потоков в программе?

ЛАБОРАТОРНАЯ РАБОТА 6. ПЕРЕДАЧА ДАННЫХ ПОТОКАМ

1. Цель и содержание

Цель лабораторной работы: научиться создавать многопоточные приложения с возможностью передачи параметров в параллельно выполняемые потоки.

Задачи лабораторной работы:

– изучить механизм передачи параметров в поток;

– научиться решать практические задачи с использованием потоков с параметрами;

– изучить возможные методы передачи параметров.

2. Теоретическое обоснование

Передавать потоку какие-то данные можно двумя способами: в конструкторе Thread с помощью делегата ParameterizedThreadStart либо за счет создания специального класса и определения метода потока как метода экземпляра, что позволит инициализировать данные экземпляра перед запуском потока.

Для передачи данных потоку необходим какой-то класс или структура, где бы можно было сохранить данные. В рассматриваемом примере определяется структура Data, которая содержит строку, но в принципе передавать можно любой объект:

```
public struct Data
{
    public string Message;
}
```

В случае применения делегата ParameterizedThreadStart точка входа в поток должна обязательно принимать параметр типа object и возвращать

void. Объект можно привести к нужному типу. Например, ниже показано, как вывести на консоль сообщение:

```
static void ThreadMainWithParameters(object o)
{
    Data d = (Data)o;
    Console.WriteLine("Выполняется в потоке, получено {0}", d.Message);
}
```

В конструкторе класса Thread можно назначить новую точку входа ThreadMainWithParameters и вызвать метод Start ( ), передав ему переменную d:

```
static void Main()
{
    var d = new Data { Message = "Info" };
    var t2 = new Thread(ThreadMainWithParameters);
    t2.Start(d);
}
```

Другим способом для передачи данных новому потоку является определение класса (например, MyThread) с необходимыми полями, а также главного метода потока как метода экземпляра этого класса:

```
public class MyThread
{
    private string data;
    public MyThread(string data)
    {
        this.data = data;
    }
    public void ThreadMain()
    {
        Console.WriteLine("Выполняется в потоке, data: {0}", data);
    }
}
```

В этом случае можно создать объект класса MyThread и передать его и метод ThreadMain ( ) конструктору класса Thread. Поток сможет получать доступ к данным:

```
static void Main()
{
    var obj = new MyThread("info");
    var t3 = new Thread(obj.ThreadMain);
}
```

3. Методика и порядок выполнения работы

1. Создайте консольное приложение в среде MS Visual Studio.

2. С помощью параметризированных потоков выполните решение задачи в соответствии с индивидуальным вариантом.

Индивидуальное задание.

Вариант	Решаемая задача (результат метода)	Входные параметры
1	Метод находит подмножество элементов массива случайных чисел, которые отличаются от заданного числа не более чем на 4.	Два параметра: размер массива, значеное число
2	Метод находит логическое значение, указывающее существует ли заданный символ в строке	Два параметра: строка и искомый символ
3	Метод находит два числа – максимальные элементы массива случайных чисел.	Два параметра: размеры массивов
4	Метод находит скадярное произведение двух случайных векторов.	Один параметр: размер векторов
5	Метод находит сумму двух матриц случайных чисел	Два параметра: размер матриц
6	Метод находит количество вхождений заданного элемента в матрице вещественных чисел	Два параметра: матрица и целевой элемент
7	Метод находит результат сложения двух случайных векторов целых чисел	Один параметр: размер векторов
8	Метод находит количество вхождений заданного символа в строке	Два параметра: строка и целевой символ
9	Метод находит подмножество четных элементов матрицы случайных чисел	Два параметра: размер матрицы
10	Метод находит статистику массива случайных целых чисел: список пар (число массива, количество вхождений)	Один параметр: размер массива
11	Метод возвращает логическое значение, указывающее существует ли заданный элемент в матрице чисел double	Два параметра: матрица и искомый элемент
12	Метод находит сумму элементов матрицы целых случайных чисел	Два параметра: размер матрицы
13	Метод находит разницу максимального и минимального элементов матрицы целых случайных чисел	Два параметра: размер матрицы
14	Метод находит логическое значение, указывающее существует ли заданное число в массиве целых случайных чисел	Два параметра: размер массива и искомый элемент
15	Метод возвращает матрицу случайных битовых значений (0 или 1), формируемую случайным образом	Два параметра: размер матрицы
16	Метод возвращает строку максимальной длины на основе переданного массива строк	Один параметр: массив (или список) строк

17	Метод возвращает подмножество элементов массива случайных чисел, которые делятся на 3	Один параметр: размер исходного массива
18	Метод находит число – количество элементов присутствующих в обоих массивах случайных чисел.	Два параметра: размеры массивов
19	Метод находит среднее арифметическое элементов матрицы случайных чисел.	Два параметра: размер матрицы
20	Метод находит результат шифрования строки: каждый исходный символ строки заменяется шифрованным символом, код которого на n больше кода исходного символа.	Два параметра: исходная строка, число сдвига n
21	Метод находит подмножество элементов массива случайных чисел, которые делятся на 6	Один параметр: размер исходного массива
22	Метод находит статистику для строки: список пар (символ строки, количество вхождений)	Один параметр: анализируемая строка
23	Метод находит разницу максимального и минимального элементов матрицы.	Два параметра: размер матрицы
24	Метод находит подмножество элементов массива случайных чисел, которые являются четными	Один параметр: размер исходного массива
25	Метод находит количество нечетных элементов в матрице случайных чисел	Два параметра: размер матрицы

4. Вопросы для защиты работы

1. Опишите возможные пути передачи параметров в поток.

2. Подумайте, можно ли передать в несколько потоков один и тот же параметр (ссылку на объект)? Ответ обоснуйте.

ЛАБОРАТОРНАЯ РАБОТА 7. ТИПЫ ПОТОКОВ. УПРАВЛЕНИЕ ПОТОКАМИ

1. Цель и содержание

Цель лабораторной работы: освоить принципы создания и управления потоками.

Задачи лабораторной работы:

– научиться создавать фоновые и приоритетные потоки;

– научиться работать с пулом потоков;

– научиться решать практические задачи с использованием пула потоков.

2. Теоретическое обоснование

2.1 Фоновые и приоритетные потоки

Процесс приложения продолжает выполняться до тех пор, пока выполняется хотя бы один приоритетный поток. Если выполняется более одного приоритетного потока, а метод Main () завершается, процесс приложения остается активным до тех пор, пока не будут завершены все приоритетные потоки.

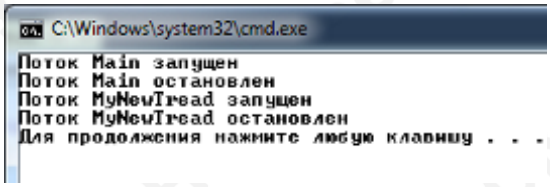
Поток, создаваемый с помощью класса Thread, по умолчанию является приоритетным. Потоки, создаваемые в пуле потоков, всегда являются фоновыми.

При создании потока с помощью класса Thread можно указать, каким должен быть поток – приоритетным или фоновым. Для этого необходимо установить свойство IsBackground.

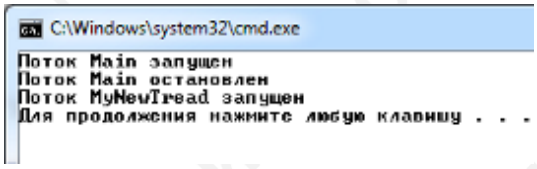
Рассмотрим пример (рис. 7.1). Вывод данной программы представлен на рис. 7.2(а).

```
static void Main(string[] args)
{
    Console.BackgroundColor = ConsoleColor.White;
    Console.Clear();
    Console.ForegroundColor = ConsoleColor.Black;

    Console.WriteLine("Поток Main запущен");

    Thread th = new Thread(
        () => {
            Console.WriteLine("Поток MyNewTread запущен");
            Thread.Sleep(3000);
            Console.WriteLine("Поток MyNewTread остановлен");
        })
        { IsBackground=false, Name ="MyNewTread" };

    th.Start();

    Console.WriteLine("Поток Main остановлен");
}
```

Рисунок 7.1 – Пример установки типа потока.



а)



б)

Рисунок 7.2 – Вывод программы: а) IsBackground=false; б) IsBackground=true.

Если изменить код программы и установить свойство потока IsBackground в значении равное true, то вывод программы изменится (рис. 7.2(б)). В первом случае поток MyNewThread является приоритетным, во втором – фоновым. Поток Main не ожидает завершения фоновых потоков.

2.2 Приоритеты потоков

За планирование потоков отвечае операционная система, но программист может влиять на этот процесс, назначая потока м приоритеты.

Операционная система планирует выполнение потоков на основе их приоритетов. Поток с наивысшим приоритетом начинает выполняться в ЦП первым. Поток прекращает выполнение и освобождает ЦП, если ему требуется ожидание какого-то ресурса. Есть несколько причин для перехода потока в режим ожидания. Например, это может происходить из-за получения команды на засыпание, из-за необходимости ожидать завершение дисковых операций ввода-вывода, поступление сетевого пакета и т.п. Если поток не освобождает ЦП самостоятельно, его вытесняет планировщик потоков. Если поток имеет выделенный квант времени, он может использовать ЦП непрерывно. Если выполняется несколько потоков с одинаковым приоритетом, каждый из которых ожидает получения доступа к ЦП, планировщик потоков применяет алгоритм кругового обслуживания, предоставляя этим потокам доступ к ЦП по очереди. В случае вытеснения поток помещается в конец очереди.

В классе Thread базовый приоритет потока устанавливается в свойстве Priority. Допустимые значения определены в перечислении ThreadPriority. Эти значения представляют различные уровни приоритета и выглядят следующим образом: Highest, AboveNormal, Normal, BelowNormal и Lowest.

При назначении потоку более высокого приоритета следует соблюдать осторожность, поскольку это уменьшает шансы на выполнение для других потоков. Если это настоятельно необходимо, то лучше изменять приоритет только на короткое время.

2.3 Состояния потоков

Поток создается за счет вызова метода Start () объекта Thread. Однако после вызова метода Start( ) новый поток все еще пребывает не в состоянии

Running, а в состоянии Unstarted. В состояние Running поток переходит сразу после того, как планировщик потоков операционной системы выберет его для выполнения. Информация о текущем состоянии потока доступна через свойство Thread.ThreadState.

С помощью метода Thread.Sleep( ) поток можно перевести в состояние WaitSleepJoin и при этом указать, через какой промежуток времени поток должен возобновить работу. Чтобы остановить поток, необходимо вызвать метод Thread.Abort( ). При вызове этого метода в соответствующем потоке генерируется исключение типа ThreadAbortException.

В случае если для этого исключения предусмотрен обработчик, перед завершением поток сможет выполнить необходимые операции по очистке. Чтобы продолжить выполнение потока после выдачи исключения ThreadAbortException, следует вызвать метод Thread.ResetAbort( ). Состояние потока, получающего запрос на немедленное прекращение, изменяется с AbortRequested на Aborted, если поток не производит сброс.

Если необходимо дожидаться завершения работы потока, можно вызвать метод Thread.Join( ). Этот метод блокирует текущий поток и переводит его в состояние WaitSleepJoin до тех пор, пока не будет завершен присоединенный к нему поток.

2.4 Пулы потоков

Создание потоков требует времени. Если есть различные короткие задачи, подлежащие выполнению, можно создать набор потоков заранее и затем просто отправлять соответствующие запросы, когда наступает очередь для их выполнения.

На практике часто требуется, чтобы в зависимости от потребности конкретной задачи количество потоков увеличивалось и уменьшалось автоматически. Реализовывать подобное поведение программисту

самостоятельно не требуется. Для управления таким поведением потоков в .NET Framework предусмотрен класс ThreadPool (класс пула потоков).

Значение максимально допустимого количества потоков в пуле может изменяться. В случае двуядерного ЦП оно по умолчанию составляет 1023 рабочих потоков и 1000 потоков ввода-вывода.

Можно указывать минимальное количество потоков, которые должны запускаться сразу после создания пула, и максимальное количество потоков, доступных в пуле. Если остались какие-то подлежащие обработке задания, а максимальное количество потоков в пуле уже достигнуто, то более новые задания будут помещаться в очередь и там ожидать, пока какой-то из потоков завершит свою работу.

Рассмотрим пример пула потоков для обработки элементов массива. Как только очередной элемент добавляется в массив, в пул потоков добавляется новая задача (для обработки добавленного элемента).

На рис. 7.3 представлен класс, который описывает элементы массива.

```
/// <summary>
/// Класс элементов массива
/// Data - целое число,
/// Fact - факториал числа Data
/// </summary>
6 references
public class MassItem
{
    public int Data;
    public long Fact;
}
```

Рисунок 7.3 – Класс-элемент массива.

Разработаем класс, который позволит хранить массив элементов, добавлять элементы типа MassItem в массив и запускать поток обработки элемента массива с целью вычисления факториала числа (рис. 7.4).

```
/// <summary>
/// Класс массива. Через него производится
/// добавление элементов в массив и запуск
/// задач в пуле потоков
/// </summary>
3 references
public class MassData
{
    private List<MassItem> massive;

    1 reference
    public MassData()
    {
        massive = new List<MassItem>(1000);
    }

    1 reference
    public void Add(MassItem newElement)
    {
        if (newElement.Data < 1) newElement.Data = 1;
        // Добавляем элемент в массив
        massive.Add(newElement);
        /*
         * Запускаем задачу обработки элемента (вычисление ыакториала).
         * Задача представлена в виде лямбда-выражения
        */
        ThreadPool.QueueUserWorkItem(MethodForThread, newElement);
    }

    /// <summary>
    /// Метод запускается при добавлении элемента в массив
    /// </summary>
    /// <param name="ob">Новый элемент</param>
    1 reference
    protected static void MethodForThread(object ob)
    {
        MassItem item = (MassItem)ob;
        int F = item.Data;
        Console.WriteLine("Поток {0}. Получен новый элемент {1}",
            Thread.CurrentThread.ManagedThreadId,
            item.Data);
        long Factorial = 1;
        while (F > 1)
        {
            Factorial *= F;

            Thread.Sleep(100); // Задержка

            Console.WriteLine("Поток {0}. Вычисление! Факториал от {1} равен {2}",
                Thread.CurrentThread.ManagedThreadId,
                item.Data,
                Factorial);
            F--;
        }

        item.Fact = Factorial;
        Console.WriteLine("Поток {0}. Вычисление завершено! Факториал от {1} равен {2}",
            Thread.CurrentThread.ManagedThreadId,
            item.Data,
            item.Fact);
    }
}
```

Рисунок 7.4 – Класс для работы с массивом.

Реализуем функцию Main( ), в которой производится добавдение 1000 случайных элементов (рис. 7.5).

```
static void Main(string[] args)
{
    Console.BackgroundColor = ConsoleColor.White;
    Console.Clear();
    Console.ForegroundColor = ConsoleColor.Black;

    int MaxWorkThread;
    int MaxIOTread;
    ThreadPool.GetMaxThreads(out MaxWorkThread, out MaxIOTread);
    Console.WriteLine("Максимальное количество рабочих потоков: {0}." +
        "Максимальное количество потоков ввода-вывода {1}",
        MaxWorkThread, MaxIOTread);

    // Создаем объект класса-массива и добавляем 1000 случайных элементов
    MassData myData = new MassData();
    Random rnd = new Random();
    for (int i = 0; i < 1000; i++)
    {
        myData.Add(new MassItem() { Data = rnd.Next(10) });
    }

    Console.ReadKey();
}
```

Рисунок 7.5 – Метод Main( ).

Вывод программы показан на рис. 7.6.

```
Максимальное количество рабочих потоков: 1023.
Максимальное количество потоков ввода-вывода 1000
Поток 6. Получен новый элемент 3
Поток 10. Получен новый элемент 2
Поток 11. Получен новый элемент 8
Поток 12. Получен новый элемент 1
Поток 12. Вычисление завершено! Факториал от 1 равен 1
Поток 12. Получен новый элемент 6
Поток 6. Вычисление! Факториал от 3 равен 3
Поток 10. Вычисление! Факториал от 2 равен 2
Поток 10. Вычисление завершено! Факториал от 2 равен 2
Поток 10. Получен новый элемент 2
Поток 12. Вычисление! Факториал от 6 равен 6
Поток 11. Вычисление! Факториал от 8 равен 8
Поток 10. Вычисление! Факториал от 2 равен 2
Поток 10. Вычисление завершено! Факториал от 2 равен 2
Поток 10. Получен новый элемент 7
Поток 6. Вычисление! Факториал от 3 равен 6
Поток 6. Вычисление завершено! Факториал от 3 равен 6
Поток 6. Получен новый элемент 7
Поток 12. Вычисление! Факториал от 6 равен 30
Поток 11. Вычисление! Факториал от 8 равен 56
Поток 10. Вычисление! Факториал от 7 равен 7
Поток 6. Вычисление! Факториал от 7 равен 7
Поток 12. Вычисление! Факториал от 6 равен 120
```

Рисунок 7.6 – Результат работы программы.

Изучите представленный код приложения. Исследуйте поведение программы при изменении времени задержки в функции вычисления факториала. Сделайте вывод: как меняется работа программы от изменения времени задержки.

3. Методика и порядок выполнения работы

1. Создайте консольное приложение.

2. Разработайте классы для решения задачи в соответствии с индивидуальным вариантом с использованием пула потоков. В каждом задании необходимо разработать коллекцию элементов заданного типа и класс для управления коллекцией, осуществляющий обработку элемента с использованием пула потоков.

3. В алгоритме решения задачи предусмотрите задержку алгоритма с использованием метода Thread.Sleep( ).

4. Сделайте вывод: как меняется поведение программы при изменении времени искусственной задержки алгоритма.

Индивидуальное задание.

Вариант	Задача, решаемая с использованем пула потоков (метод обработки элемента)	Тип элемента коллекции
1	Метод находит среднее арифметическое элементов матрицы случайных чисел.	Класс определяет матрицу (размер выбирается случайным образом)
2	Метод находит результат шифрования строки: каждый исходный символ строки заменяется шифрованным символом, код которого на n больше кода исходного символа.	Класс представляет пару строк: исходная и шифрованная (исходная задается в конструкторе)
3	Метод находит подмножество элементов массива случайных чисел, которые делятся на 6	Класс представляет массив случайных чисел, случайного размера
4	Метод находит статистику для строки: список пар (символ строки, количество вхождений)	Класс представляет строку (задается в конструкторе)
5	Метод находит разницу максимального и минимального элементов матрицы	Класс представляет матрицу случайных чисел случайного размера

6	Метод находит подмножество элементов массива случайных чисел, которые являются четными	Класс представляет массив случайных чисел случайного размера
7	Метод находит количество нечетных элементов в матрице случайных чисел	Класс представляет матрицу случайных чисел (размер задается в конструкторе)
8	Метод находит разницу максимального и минимального элементов матрицы целых случайных чисел	Класс описывает матрицу случайных чисел (размер матрицы выбирается случайным образом)
9	Метод находит логическое значение, указывающее существует ли заданное число в массиве целых случайных чисел	Класс описывает массив случайных чисел (размер массива выбирается случайным образом, искомое число указывается в конструкторе)
10	Метод строит матрицу случайных битовых значений (0 или 1), формируемую случайным образом	Класс описывает матрицу (размер указывается в конструкторе)
11	Метод находит символ строки, который встечается максимальное количество раз	Класс описывает строку (строка задается в конструкторе)
12	Метод находит подмножество элементов массива случайных чисел, которые делятся на 3	Класс описывает массив
13	Метод находит число – количество элементов присутствующих в обоих массивах случайных чисел.	Класс описывает пару массивов (размер определяется случайным образом, элементы массива – случайные числа)
14	Метод находит подмножество элементов массива случайных чисел, которые отличаются от заданного числа не более чем на 4.	Класс для описания массива случайных чисел и заданного числа
15	Метод находит логическое значение, указывающее существует ли заданный символ в строке	Класс представляет строку и искомый символ
16	Метод находит два числа – максимальные элементы массива случайных чисел.	Класс для описания массива случайных элементов

17	Метод находит скадярное произведение двух случайных векторов.	Класс описывает два случайных вектора размерности n.
18	Метод находит сумму двух матриц случайных чисел	Класс описывает пару матриц случайного размера, содержащих случайные числа
19	Метод находит количество вхождений заданного элемента в матрице вещественных чисел	Класс описывает случайную матрицу и искомое число
20	Метод находит результат сложения двух случайных векторов целых чисел	Класс описывает пару случайных векторов размера n (n задается в конструкторе)
21	Метод находит количество вхождений заданного символа в строке	Класс представляет строку и искомое число
22	Метод находит подмножество четных элементов матрицы случайных чисел	Класс описывает матрицу случайных чисел (размер матрицы указывается в конструкторе)
23	Метод находит статистику массива случайных целых чисел: список пар (число массива, количество вхождений)	Класс описывает матрицу случайных чисел (размер матрицы выбирается случайным образом)
24	Метод находит результат – логическое значение, указывающее существует ли заданный элемент в матрице чисел double	Класс описывает матрицу случайных чисел (размер матрицы указывается в конструкторе)
25	Метод находит сумму элементов матрицы целых случайных чисел	Класс описывает матрицу случайных чисел (размер матрицы выбирается случайным образом)

4. Вопросы для защиты работы

1. Какие типы потоков вы знаете? Чем они отличаются?

2. Для чего применяются приоритеты потоков? Как задать приоритет потока?

3. Что такое пул потоков? Какой метод запускает поток в пуле?

ЛАБОРАТОРНАЯ РАБОТА 8. СОЗДАНИЕ И ЗАПУСК ЗАДАЧ

1. Цель и содержание

Цель лабораторной работы: изучить механизм работы с классами задач.

Задачи лабораторной работы:

– изучить возможности класса Task;

– научиться создавать и запускать задачи;

– научиться работать с массивами задач.

2. Теоретическое обоснование

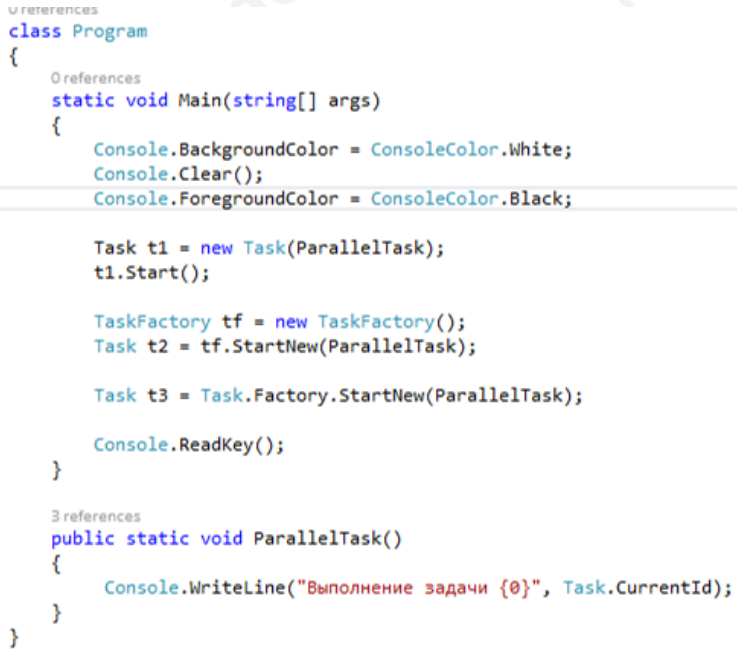
В состав .NET 4 входит новое пространство имен System.Threading. Tasks, в котором содержатся классы для абстрагирования функциональности потоков. Внутри этих классов используется ThreadPool. Под задачей (task) понимают единицу подлежащей выполнению работы. Эта единица работы может выполняться в отдельном потоке, но может также запускаться в синхронизированной манере, ожидая вызывающего потока. В случае применения задачи создается дополнительный уровень абстракции и также появляется масса возможностей для управления лежащими в основе потоками.

Задачи обеспечивают большую гибкость в плане организации выполняемой работы.

Для запуска задачи можно использовать либо класс TaskFactory, либо конструктор класса Task и метод Start( ). Конструктор Task просто предоставляет больше гибкости при создании задачи.

При запуске задачи можно создать экземпляр класса Task и назначить выполняемый код с использованием делегата Action или Action<object> (т.е. без параметров либо с одним параметром типа object).

В представленном листинге показаны варианты запуска задач (рис. 8.1).



```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Console.BackgroundColor = ConsoleColor.White;
        Console.Clear();
        Console.ForegroundColor = ConsoleColor.Black;

        Task t1 = new Task(ParallelTask);
        t1.Start();

        TaskFactory tf = new TaskFactory();
        Task t2 = tf.StartNew(ParallelTask);

        Task t3 = Task.Factory.StartNew(ParallelTask);

        Console.ReadKey();
    }

    3 references
    public static void ParallelTask()
    {
        Console.WriteLine("Выполнение задачи {0}", Task.CurrentId);
    }
}
```

Рисунок 8.1 – Варианты запуска задач.

Первый способ запуска задач предполагает создание объекта типа Task и запуск задачи методом Start( ). До вызова Start( ) задача находится в состоянии Created, то есть задача не запускается автоматически после создания. Данный подход похож на использование асинхронных делегатов – задача запускается в асинхронном режиме. Задачу можно запустить в синхронном (непараллельном) режиме; для этого необходимо использовать метод RunSynchronously( ) вместо Start( ).

Второй способ предполагает использование объекта TaskFactory: создание объекта и вызов метода StartNew( ) с параметром-делегатом.

Третий способ – вызов метода StartNew( ) через свойство Factory класса Task.

Конструктору Task и методу StartNew( ) класса TaskFactory можно передавать значения из перечисления TaskCreationOptions. За счет передачи значения LongRunning можно информировать планировщик задач о том, что

выполнение задачи требует длительного времени, чтобы он мог по возможности выделить для нее новый поток.

Если задача должна быть присоединена к родительской задаче и, следовательно, отменяться в случае отмены этой родительской задачи, необходимо использовать значение AttachToParent. Устанавливая значение PreferFairness, можно указывать планировщику, что он сначала должен брать задачи, ожидающие выполнения. По умолчанию так не происходит, если задача создается в рамках другой задачи. Если задачи предусматривают создание дополнительных единиц работы за счет использования дочерних задач, то эти дочерние задачи получают более высокий приоритет по сравнению с другими задачами.

Они не дожидаются завершения работы в очереди пула потоков. Если такие задачи должны обрабатываться равноправно со всеми остальными задачами, понадобится установить значение PreferFairness.

3. Методика и порядок выполнения работы

1. Создайте консольное приложение в среде Visual Studio.

2. Реализуйте решение задачи в соответствии с индивидуальным вариантом с применением класса Task.

Индивидуальное задание.

Вариант	Метод для реализации
1	Метод расчета суммы элементов массива случайных чисел.
2	Метод преобразования матрицы случайных чисел: элементы кратные 3 заменяются на квадрат соответствующего элемента.
3	Метод вычисления факториала числа.
4	Метод преобразования матрицы случайных чисел: каждый элемент заменяется синусом элемента матрицы.
5	Метод преобразования матрицы случайных чисел: нечетные элементы заменяются –1.
6	Метод поиска среднего арифметического элементов матрицы.

7	Метод вычисления минимального значения в матрице случайных чисел.
8	Метод преобразования матрицы случайных чисел: четные элементы заменяются суммой косинуса и синуса соответствующего элемента матрицы.
9	Метод преобразования матрицы случайных чисел: нечетные элементы заменяются суммой тангенса и котангенса соответствующего элемента матрицы.
10	Метод вычисления статистики по случайной матрице (количество элементов для каждого значения матрицы).
11	Метод вычисления среднего арифметического элементов массива.
12	Метод поиска максимального элемента в матрице.
13	Метод преобразования матрицы случайных чисел: каждый элемент заменяется косинусом элемента матрицы.
14	Метод подсчета количества четных элементов в матрице.
15	Метод вычисления скалярного произведения двух случайных векторов.
16	Метод преобразования матрицы случайных чисел: все элементы заменяются суммой косинуса и синуса соответствующего элемента матрицы.
17	Метод вычисления среднего арифметического трехмерного массива случайных чисел.
18	Метод вычисления суммы двух матриц.
19	Метод расчета модуля случайного вектора.
20	Метод преобразования матрицы случайных чисел: четные элементы заменяются тангенсом элемента матрицы.
21	Метод подсчета количества чисел, кратных 3, в массиве случайных чисел.
22	Метод вычисления статистики (количество для каждого входящего символа) по строке.
23	Метод вычисления статистики по случайному массиву (количество элементов для каждого значения массива).
24	Метод подсчета количества четных и нечетных элементов матрицы.
25	Метод вычисления статистики по трехмерному массиву (количество элементов для каждого значения массива).

4. Вопросы для защиты работы

1. Что такое класс Task? Чем класс Task отличается от класса Thread?

2. Как получить идентификатор текущей задачи? Как получить идентификатор текущего потока?

3. Опишите возможные методы создания и запуска задач.

4. Какие параметры задачи может установить программист при реализации задачи? Опишите возможные варианты применения параметра типа TaskCreationOptions.

ЛАБОРАТОРНАЯ РАБОТА 9. ЗАДАЧИ ПРОДОЛЖЕНИЯ

1. Цель и содержание

Цель лабораторной работы: научиться использовать задачи продолжения в многопоточных программах.

Задачи лабораторной работы:

– научиться формирвоать последовательность запуска задач;

– научиться использовать задачи продолжения для формирования последовательности выполняемых работ;

– научиться использовать задачи продолжения для реализации массового запуска задач.

2. Теоретическое обоснование

В случае применения задач можно указывать, что после завершения одной задачи должна начинаться какая-то другая задача. В данном механизме можно также реализовать условное выполнение: реализовать запуск определенной задачи продолжения в зависимости от характера завершения основной задачи (плановое завершение, ошибка и т.д.).

Задача продолжения принимает параметр типа Task, который позволяет получать информацию об исходной задаче. Задача продолжения определяется вызовом метода ContinueWith( ) объекта исходной задачи. Развитыми средствами определения и запуска задач продолжения обладает также класс TaskFactory.

Рассморим пример использования ContinueWith( ) (рис. 9.1).

```
static void Main(string[] args)
{
    Task tBase = new Task(
        () => {
            Console.WriteLine("Основная задача. Начало");
            Thread.Sleep(3000);
            Console.WriteLine("Основная задача. Окончание");
        });

    Task tContinue = tBase.ContinueWith(
        base_task => {
            Console.WriteLine("Задача {0} завершена.", base_task.Id);
            Console.WriteLine("Задача продолжения. Начало");
            Thread.Sleep(3000);
            Console.WriteLine("Задача продолжения. Окончание");
        });

    tBase.Start();

    Console.ReadKey();
}
```

Рисунок 9.1 – Использование ContinueWith( ) для определения задачи продолжения.

Очевидно, что использовать ContinueWith( ) можно для определения нескольких задач продолжения. Причем каждая задача продолжения может иметь несколько своих задач продолжения. Таким образом, можно строить сложные иерархии задач. В качестве второго параметра метода ContinueWith( ) можно указать значение из перечисления TaskContinuationOptions. Каждое из значений (OnlyOnFaulted, NotOnFaulted, OnlyOnCanceled, NotOnCanceled, OnlyOnRanToCompletion и пр.) указывает на условие запуска задачи продолжения.

3. Методика и порядок выполнения работы

1. Создайте консольное приложение.

2. Реализуйте решение задачи с использованием механизма задач продолжения в соответствии с вариантом индивидуального задания.

Индивидуальное задание.

Вариант	Основная задача	Задачи продолжения
1	Генерация матрицы случайных чисел (размер задается пользователем)	2 задачи: расчет суммы элементов; поиск максимального элемента.
2	Генерация массива случайных чисел (размер задается пользователем)	2 задачи: вычисление количества элементов, делящихся на 3; поиск минимального элемента.
3	Генерация матрицы случайных чисел (размер определяется случайным образом)	2 задачи: расчет суммы всех четных элементов; поиск среднего арифметического элементов.
4	Генерация массива случайных чисел (размер определяется случайным образом)	2 задачи: расчет суммы нечетных элементов; подсчет количества элементов, кратных 3.
5	Генерация матрицы случайных чисел (размер задается пользователем)	2 задачи: вывод матрицы в консоль; подсчет количества элементов, кратных 7.
6	Генерация массива случайных чисел (размер задается пользователем)	2 задачи: расчет суммы квадратов четных элементов; поиск максимального элемента.
7	Генерация матрицы случайных чисел (размер определяется случайным образом)	2 задачи: поиск максимального элемента в каждом столбце; поиск минимального элемента матрицы.
8	Генерация массива случайных чисел (размер определяется случайным образом)	3 задачи: расчет суммы кубов нечетных элементов; поиск максимального элемента; поиск минимального элемента.
9	Генерация матрицы случайных чисел (размер задается пользователем)	2 задачи: поиск минимального элемента в каждом столбце; поиск элементов матрицы, отличающихся не более чем на 4 от заданного пользователем.
10	Генерация массива случайных чисел (размер задается пользователем)	2 задачи: поиск максимального элемента; подсчет количества элементов массива, делящихся на 5.

11	Генерация матрицы случайных чисел (размер определяется случайным образом)	3 задачи: вывод матрицы в консоль; подсчет количества элементов массива, которые делятся на 3; поиск минимума в каждой строке.
12	Генерация массива случайных чисел (размер определяется случайным образом)	2 задачи: поиск минимального элемента среди нечетных; подсчет количества элементов массива, делящихся на 7.
13	Генерация матрицы случайных чисел (размер задается пользователем)	3 задачи: расчет суммы кубов четных элементов; поиск среднего арифметического элементов; поиск минимального элемента.
14	Генерация массива случайных чисел (размер задается пользователем)	2 задачи: поиск максимального элемента среди четных; подсчет количества элементов массива, отличающихся от заданного пользователем не более чем на 3.
15	Генерация матрицы случайных чисел (размер определяется случайным образом)	3 задачи: расчет суммы четных элементов, делящихся на 4; поиск максимума среди элементов, делящихся на 3; поиск минимального элемента.
16	Генерация массива случайных чисел (размер определяется случайным образом)	3 задачи: вычисление квадрата суммы элементов; вычисление суммы кубов элементов; вывод массива в консоль.
17	Генерация матрицы случайных чисел (размер задается пользователем)	3 задачи: расчет суммы элементов, делящихся на 4; расчет количества элементов, делящихся на 3; поиск минимального элемента.
18	Генерация массива случайных чисел (размер задается пользователем)	3 задачи: расчет суммы нечетных элементов, делящихся на 7; поиск максиального элемента; вывод массива в консоль.

19	Генерация матрицы случайных чисел (размер определяется случайным образом)	3 задачи: расчет суммы нечетных элементов, делящихся на 7; поиск максимума среди элементов, делящихся на 2; вывод матрицы в консоль.
20	Генерация массива случайных чисел (размер определяется случайным образом)	3 задачи: расчет суммы четных элементов, делящихся на 4; поиск максимума среди элементов, делящихся на 3; поиск минимального элемента.
21	Генерация матрицы случайных чисел (размер задается пользователем)	3 задачи: расчет суммы кубов элементов, делящихся на 3; поиск максимального элемента; поиск минимального элемента в каждом столбце.
22	Генерация массива случайных чисел (размер задается пользователем)	3 задачи: расчет суммы нечетных элементов, делящихся на 5; поиск максимума среди элементов, делящихся на 9; поиск минимального элемента среди четных.
23	Генерация матрицы случайных чисел (размер определяется случайным образом)	3 задачи: расчет суммы элементов, делящихся на 5; поиск максимума среди элементов, делящихся на 9; вывод матрицы в консоль.
24	Генерация массива случайных чисел (размер определяется случайным образом)	3 задачи: расчет суммы квадратов четных элементов; поиск максимума среди элементов, делящихся на 9; вывод всех элементов массива.
25	Генерация матрицы случайных чисел (размер задается пользователем)	3 задачи: расчет суммы нечетных элементов, делящихся на 2; поиск максимума в каждом столбце; поиск минимального элемента в каждой строке.

4. Вопросы для защиты работы

1. Что такое задача продолжения? Для чего используется данный механизм?

2. Сколько задач продолжения может иметь каждая задача?

3. С помощью какого метода класса Task можно задать задачу продолжения?

4. Какие параметры принимает метод, представляющий задачу продолжения?

5. Поясните механизм определения условных задач продолжения.

6. Опишите методы класса TaskFactory, предназначенные для указания задач продолжения.

7. Опишите механизм, использующий задачи продолжения и позволяющий запустить большое количество задач (100, 1000, …) одной командой Start( ).

СПИСОК ЛИТЕРАТУРЫ

1. Вилле К. Представляем C# / Вилле К. – М.: ДМК Пресс, 2008. – 183 с. Режим доступа: http://e.lanbook.com.

2. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. – М.: ДМК Пресс, 2007. – 368 с. Режим доступа: http://e.lanbook.com.

3. Рандольф, Н. Visual Studio 2010 для профессионалов / Н. Рандольф, Д. Гарднер. – М.: Диалектика, 2011. – 1184 с. – ISBN 978-5-8459-1683-9.

4. Уотсон, К. Visual C# 2010. Полный курс / К. Уотсон, К. Нагел. – М.: Вильямс, 2010. – 960 с. – ISBN 978-5-8459-1699-0, 978-0-470-50226-6.

5. Учебное пособие «Технологии программирования I» для студентов специальности 09.03.02 «Информационные системы и технологии» / сост. Николаев Е.И.; рец. Мочалов В.П, Маликов А.В. – Ставрополь : СКФУ, 2011. – 150 с.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ по дисциплине

«Введение в технологии высокороизводительных вычислений»

для студентов специальности

09.03.02 «Информационные системы и технологии»

Составитель: Е.И. Николаев