МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО НАПИСАНИЮ САМОСТОЯТЕЛЬНЫХ РАБОТ
ПО ДИСЦИПЛИНЕ «Технологии параллельного программирования»

Для студентов заочного отделения, обучающихся
по специальности: 230100.62 – Информатика и вычислительная техника

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

В соответствии с учебным планом студенты заочного отделения выполняют контрольную работу, состоящую из теоретических вопросов и решения задачи. Задания по контрольным работам построены таким образом, чтобы выяснить подготовку студента по различным разделам изучаемой дисциплины.

Целью настоящих методических рекомендаций является оказание практической помощи студенту в выполнении контрольной работы. Выполнить контрольную работу следует после ознакомления со всеми темами дисциплины. Темы, вынесенные в задание, требуют от студента углубленного изучения.

Ответ на каждый вопрос должен излагаться подробно, с разъяснениями, со ссылкой на нормативно-правовые акты.

К каждому варианту прилагается список литературы. По своей инициативе студент может использовать другие дополнительные источники.

Контрольная работа выполняется в электронном виде в формате текстового процессора Word. Вопросы задания и текст задачи должен быть перепечатаны студентом с указанием варианта. Обязательной является ссылка на соответствующие статьи нормативных актов.

Контрольная работа выполняется по вариантам. Каждый вариант может содержать несколько заданий по конкретной тематике. Тематику определяет вариант. Студент получает в качестве задания на контрольную работу номер

варианта задания и, если заданий в варианте несколько, то и номер задания из варианта.

Независимо от варианта, работа включает следующие обязательные разделы:

1. Описание метода решения задачи.
2. Обоснование выбранного метода.
3. Листинг программы.
4. Описание результатов тестирования.

В качестве инструментария разработчика ПО следует использовать транслятор языка, поддерживающий программирование многопоточных приложений. Например, C++, Java, C# и F#.

Первый лист работы – титульный. В нем указывается наименование образовательного учреждения, дисциплина, вариант, фамилия, имя и отчество студента, факультет, курс, группа.

Контрольная работа представляется до начала экзаменационной сессии.

Теоретическая часть

1 Параллельные алгоритмы

На практике чаще всего приходиться разрабатывать программы, реализующие последовательный алгоритм.

Определение 1. Алгоритм называется последовательным, если все его действия выполняются в том порядке, в котором они указаны.

Из данного определения следует, что в последовательном алгоритме имеется явная зависимость каждого последующего шага от предыдущего.

Определение 2. Алгоритм, часть действий которого может выполняться одновременно (параллельно) несколькими исполнителями, называется параллельным.

Отсюда следует, что процессы создания программ, реализующих последовательный и параллельный алгоритмы, называются соответственно *последовательным и параллельным программированием*

2 Параллельные процессы

Из технологий программирования известно, что в результатом компиляции любой программы, написанной на языке программирования, является *загрузочный модуль* (*виртуальная машина, программа*), который реализует конкретный алгоритм. В технологии .NET аналогом виртуальной машины является сборка (assembly).

Известно, также, что любой загрузочный модуль, исполняемый компьютером, называется *задачей*. Если задача реализует последовательный алгоритм, то её выполняет один центральный процессор компьютера или его ядро. В этом случае говорят, что задачу выполняет один *процесс*. При этом саму задачу называют процессом.

В случае параллельного алгоритма, те его действия, которые выполняются одновременно, реализуются отдельными процессорами или их ядрами, а сами действия называются параллельными процессами.

Параллельные процессы могут полностью не зависеть друг от друга, либо периодически синхронизироваться и взаимодействовать друг с другом.

Процессы удобно изображать в виде направленных графов, чьи дуги помечены именами процессов, а стрелки указывают на порядок выполнения действий, реализующих алгоритм.

Параллельные процессы, которые могут быть разбиты только на последовательные и параллельные, называются *последовательно-параллельными*.

3 Примеры многопоточных приложений

В следующих разделах представлены примеры программ, реализующих многопоточность. В качестве рабочего языка программирования выбран язык C#. Перед изучением примеров и началом выполнения контрольной работы,

рекомендуется изучить методы и приёмы программирования параллельных вычислений, например по источникам [1, 2].

3.1 Вывод целых чисел, кратных заданному числу

```
using System;
using System.Threading;

//Класс исходных данных
class Data
{
   //левая граница массива:
   public long s { get; set; }

   //правая граница массива:
   public long e { get; set; }

   //кратность:
   public long k { get; set; }
}

class Program
{
   static public void Main()
   {
      //объявление переменной потока:
      Thread t = new Thread (func);
      //объявление экземпляра класса исходных данных:
      Data d = new Data ();
      //инициализация исходных данных:
      d.s = 1;
      d.e = 1000;
      d.k = 5;

      //старт вычислений в фоновом потоке с передачей исходных данных:
      t.Start (d);

      //старт вычислений в приоритетном потоке:
```

```
        Searcher (d.s + d.e, 2 * d.e, d.k);
    }

    //статический метод вывода чисел, кратных k:
    static void Searcher(long s, long e, long k)
    {
        long i;
        for (i = s; i < e; i++) {
            if(i % k == 0)
                Console.WriteLine (i);
            Thread.Sleep (0);
        }
    }

    //статический метод, выполняемый фоновым потоком:
    static void func(object t)
    {
        long s = ((Data)t).s;
        long e = ((Data)t).e;
        long k = ((Data)t).k;

        Searcher (s, e, k);
    }
}
```

3.2 Поиск максимального значения

```
using System;
using System.Threading;


//класс для передачи исходных данных в поток и извлечения результата:
class Data
{
```

```
        //массив с исходными данными:
        public int[] arr;
        //переменная для результата вычислений:
        public int inam;
}

class Program
{
        static void Main()
        {
                //ввод числа элементов массива:
                Console.Write ("n = ");
                int n = Convert.ToInt16(Console.ReadLine ());
                int i;
                //генерация массива исходных данных:
                int[] a = GetRNDArray (n);
                for(i = 0; i < n; i++)
                        Console.Write (a[i] + ", ");
                Console.WriteLine ();

                int n1 = n / 2;
                int n2 = n - n1;
                //массив с исходными данными для обработки главным потоком:
                int[] arr1 = new int[n1];
                for(i = 0; i < n1; i++)
```

```
        arr1[i] = a[i];
//массив с исходными данными для обработки второстепенным
//потоком:
int[] arr2 = new int[n2];
for(i = 0; i < n2; i++)
        arr2[i] = a[i + n1];
//экземпляр класса для исходных данных и результата:
Data data = new Data();

//переменная второстепенного потока:
Thread th = new Thread(func);

//инициализация переменной исходными данными для
//второстепеного потока:
data.arr = arr2;
//запуск второстепенного потока:
th.Start (data);

//вычисление максимального значения главным потоком:
int m1 = GetMax(arr1, "главный поток");

//ожидание завершения второстепенного потока:
while(data.inam == 0);

//вычисление макимального значения в двух массивах:
```

```
        if(m1 < data.inam)
                m1 = data.inam;

        //вывод результата:
        Console.WriteLine ("Максимальное значение: " + m1);
}

//функция, реализующая вычисления во втором потоке:
static void func(object obj)
{
        Data d = (Data)obj;
        d.inam = GetMax(d.arr, "второстепенный поток");
}

//функция вычисления максимального значения массива:
static int GetMax(int[] a, string name)
{
        int max = a[0];
        int n = a.Length;

        for(int i = 1; i < n; i++)
        {
                Console.WriteLine ("выполняется " + name);
                if(max < a[i]) max = a[i];
                Thread.Sleep (0);
```

```
            }

            return max;
        }

        //генерация массива n случайных чисел:
        static int[] GetRNDArray(int n)
        {
            int[] a = new int[n];
            Random rnd = new Random(DateTime.Now.Millisecond);

            for(int i = 0; i < n; i++)
                //получение очередного целого неотрицательного числа, не
                //превышающего 100:
                a[i] = rnd.Next(100);
            return a;
        }
    }
```

Контрольные работы

С помощью параллельного программирования решить следующие задачи:

Вариант 1. Найти сумму элементов числового массива.

Вариант 2. Найти произведение элементов числового массива.

Вариант 3. Построить вектор, элементы которого равны сумме соответствующих элементов двух других векторов.

Вариант 4. Построить вектор, элементы которого равны произведению соответствующих элементов двух других векторов.

Вариант 5. Вычислить скалярное произведение векторов.

Вариант 6. Вычислить длину вектора.

Вариант 7. Вычислить дисперсию (среднее квадратичное отклонение) результатов испытаний.

Вариант 8. Вычислить математическое ожидание дискретной случайной величины, зная закон её распределения.

Вариант 9. Определить центр масс многоугольника.

Вариант 10. Вычислить периметр многоугольника.

Вариант 11. Вычислить расстояние между двумя векторами.

Вариант 12. Вычислить среднее значение элементов числового массива.

Вариант 13. Вычислить максимальное и минимальное значения числового массива.

Вариант 14. Транспонировать квадратную матрицу.

Вариант 15. Найти максимальное и минимальное значения прямоугольной матрицы.

Вариант 16. Найти сумму диагональных элементов и сумму всех элементов квадратной матрицы.

Вариант 17. Удалить элементы числового массива, превосходящие заданное число.

Вариант 18. Удалить повторяющиеся элементы прямоугольной матрицы, превосходящие заданное число.

Вариант 19. Нормировать вектор с помощью евклидовой нормы: $\|x\|_2 = \sqrt{\sum_i |x_i|^2}$.

Вариант 20. Нормировать вектор с помощью нормы: $\|x\|_1 = \sum_i |x_i|$.

Вариант 21. Нормировать вектор с помощью нормы: $\|x\|_\infty = \max|x_i|$.

Вариант 22. Нормировать матрицу с помощью m-нормы: $\|A\|_m = \max_i \sum_j |a_{ij}|$.

Вариант 23. Нормировать матрицу с помощью l-нормы: $\|A\|_l = \max_j \sum_i |a_{ij}|$.

Вариант 24. Нормировать матрицу с помощью нормы Фробениуса: $\|A\|_2 = \sqrt{\sum_{ij} |a_{ij}|^2}$.

Вариант 25. Вычислить произведение прямоугольной матрицы на вектор-столбец.

Вариант 26. Вычислить произведение двух прямоугольных матриц.

Вариант 27. Сложить две прямоугольные матрицы.

Вариант 28. Вычислить произведение вектор-строки на прямоугольную матрицу.

Библиографический список

Основной

1. Нейгел К., Ивьен Б., Глинн Дж., Уотсон К., Скиннер М. C# 4.0 и платформа .NET 4 для профессионалов. М:. И.Д. «Вильямс», 2011. – 1440 с.
2. Смит К. Программирование на F#. – СПб.: Символ-Плюс, 2011. – 448 с

Дополнительный

3. Троелсен, Э. Язык программирования C# 2010 и платформа .NET 4.0. – М:.Издательский дом «Вильямс», 2011. – 1392 с.
4. Липаев В.В. Проектирование программных средств: Учеб. пособие для вузов по спец. Автом. сист. обр. информ. и упр.. – М.: Высш. шк., 1990. – 303 с.: ил.
5. Мирошниченко Е.А. Технология программирования: Учеб. пособие. – Томск: Изд-во ТПУ, 1999. – 88 с.
6. Иванова Г.С. Технология программирования: Учебник для вузов. – М.: Изд-во МГТУ им. Н.Э.Баумана, 2002