ГБОУ ВПО Нижегородский государственный технический университет им. Р. Е. Алексеева
Институт радиоэлектроники и информационных технологий, кафедра "Вычислительные системы и технологии"

СОГЛАСОВАНО

Доцент каф. ВСТ

_____ *Гай В. Е.*

"____" _____ _____

ТЕХНОЛОГИИ РАСПРЕДЕЛЁННОЙ ОБРАБОТКИ ДАННЫХ

Отчет к лабораторной работе №2

РАЗРАБОТКА РАСПРЕДЕЛЁННОЙ СИСТЕМЫ ОБРАБОТКИ ДАННЫХ

Студент гр. 13-В-2

_____ *Бобко С. С.*

"____" _____ _____

2015

СОДЕРЖАНИЕ

1 ТРЕБОВАНИЯ К РАБОТЕ

Разработанный программный комплекс должен состоять из Сервера и Клиента. Функции сервера: хранение удалённого объекта, предоставляющего доступ к заданиям для обработки и результату обработки. Предусмотреть на сервере возможность одновременного доступа к критической секции кода нескольких клиентов. Критическая секция кода - та, к которой гипотетически одновременно могут обратиться несколько клиентов.

Функции клиента (на сервере хранится список клиентов - эта функция уже предусмотрена исходным кодом библиотеки RemoteBase):

а) Управляющие функции (выполняет только один клиент из всего множества клиентов, выполнение данной функции должно выполняться через вызов методов удалённого объекта (удалённый объект хранится на сервере)):

- Формирование и ведение списка заданий (под ведением понимается удаление уже обработанных и предоставление клиенту задания по запросу);

- Получение, объединение и вывод результатов вычислений (результаты вычислений должны выводиться в каждом клиенте, для этого необходимо проверять окончание обработки всех данных по таймеру; объединение результатов вычисление также можно реализовать с использованием таймера);

- Устанавливает флаг того, что управляющий клиент назначен, на сервере сохраняется идентификатор клиента;

б) Вычислительные функции

- Запрос задания с сервера (клиент должен запросить задание только после того, как эти задания были сформированы);

- Обработка данных;

- Отправка результатов обработки на сервер.

2 ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

2.1 Вариант задания

Вариант 13: Разработать алгоритм сортировки массива чисел методом пузырька

2.2 Листинг программы

2.2.1 Сервер

```
  using    System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using SortLibrary;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

namespace SortServer
{
    class Server
    {
        TcpChannel channel;
        public void Start()
        {
            channel = new TcpChannel(8081);
            ChannelServices.RegisterChannel(channel, false);
            RemotingConfiguration.RegisterWellKnownServiceType(typeof(
               SharedObject), "DataPool", WellKnownObjectMode.
               Singleton);
```

```
            Log.Print("Server has started");
        }
        public void Stop()
        {
            ChannelServices.UnregisterChannel(channel);
            Log.Print("Server has stopped");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Server srv = new Server();
            srv.Start();
            Console.In.ReadLine();
            srv.Stop();
        }
    }
}
```

2.2.2 Клиент

```
  using   System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using SortLibrary;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

namespace SortClient
{
    class Shell
    {
        TcpChannel chan;
        SharedObject obj;
        int[] arr;
        int max;
        int min;
```

```
Task task;

public Shell()
{
    chan = new TcpChannel();
    ChannelServices.RegisterChannel(chan, false);
    obj = (SharedObject)Activator.GetObject(typeof(SortLibrary
       .SharedObject), "tcp://localhost:8081/DataPool");
}

public int sort()
{
    //try
    //{
        task = obj.GetTask();
        if (task == null)
            return 0;

        arr = obj.FetchData(task);

        Console.Out.WriteLine("Полученные данные:");
        display();
        int  outer;

        for (outer = 0; outer < task.stop - task.start; outer
           ++)
        {
            for (int j = outer + 1; j < task.stop - task.start
               ; j++)
            {
                if (arr[j] < arr[outer])
                {
                    var temp = arr[outer];
                    arr[outer] = arr[j];
                    arr[j] = temp;
                }
            }
        }

        Console.Out.WriteLine("Обработанные данные:");
        display();
```

```
                obj.Finish(task, arr);
            //}
            //catch (System.Net.WebException e)
            //{
            //    Console.Out.WriteLine("Error " + e.Message);
            //}
            // task.stop = 10; //МЕГАКОСТЫЛЬ
            //task.start = 6;
            return 1;
        }
        void display()
        {
            for (int i = 0; i < task.stop - task.start; i++)
            {
                Console.Out.Write(arr[i]);
                Console.Out.Write("  ");
            }
            Console.Out.WriteLine();
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Shell shellObj = new Shell();
            Console.Out.WriteLine("Клиент запущен");

            while (shellObj.sort() != 0)
                Console.In.ReadLine();
            Console.Out.WriteLine("Задачи кончились, нажмите Enter..."
               );
            Console.ReadLine();
        }
    }
}
```

2.2.3 Библиотека

```
  using    System;
using System.Collections.Generic;

namespace SortLibrary
{
    public class SharedObject : MarshalByRefObject
    {
        static int number;
        int minim = 10000;
        int maxim = 0;

        public List<int> maxes = new List<int>();
        public List<int> mins = new List<int>();

        const int dataCount = 100; // Кол-во элементов в массиве
        const int tasksCount = 2; // максимальное кол-во задач

        Queue<Task> pendingTasks; // очередь задач ожидающих обработки
        Object tasksLock;

        //List<Task> finishedTasks;

        int[] dataArray;
        int[] MinMax;
        Object dataLock;

        public SharedObject()
        {
            Log.Print("Create tasks and data");
            //dataArray = new int[dataCount];
            pendingTasks = new Queue<Task>();
            GenerateData();
            GenerateTasks();

            tasksLock = new Object();
            dataLock = new Object();
        }
```

```
void GenerateTasks()
{
    Task temp;
    int step = dataCount / tasksCount; // на каждую задачу при
       ходится равная порция массива
    for (int i = 0; i < tasksCount; i++)
    {
        temp = new Task();
        temp.start = i * step;
        temp.stop = temp.start + step - 1;
        pendingTasks.Enqueue(temp);
    }

    //int k1 = 0;
    //int k2 = 5;

    //for (int j=0; j<2; j++)
    //{
    //    temp = new Task();
    //    for (int i = k1; i < k2; i++)
    //    {
    //        temp.indexes.Add(i);
    //    }
    //    pendingTasks.Enqueue(temp);
    //    k1 += 5;
    //    k2 += 5;
    //}
}
void GenerateData()
{
    Random r = new Random();
    dataArray = new int[dataCount];

    for (int i = 0; i < dataCount; i++)
        dataArray[i] = r.Next(0, dataCount * tasksCount);
}
static int[] Sort(int[] buff)
{
    //проверка длинны массива
    //если длина равна 1, то возвращаем массив,
```

```
        //так как он не нуждается в сортировке
        if (buff.Length > 1)
        {
            //массивы для хранения половинок входящего буфера
            int[] left = new int[buff.Length / 2];
            //для проверки ошибки некорректного разбиения массива,
            //в случае если длина непарное число
            int[] right = new int[buff.Length - left.Length];
            //заполнение субмассивов данными из входящего массива
            for (int i = 0; i < left.Length; i++)
            {
                left[i] = buff[i];
            }
            for (int i = 0; i < right.Length; i++)
            {
                right[i] = buff[left.Length + i];
            }
            //если длина субмассивов больше еденици,
            //то мы повторно (рекурсивно) вызываем функцию разбиен
               ия массива
            if (left.Length > 1)
                left = Sort(left);
            if (right.Length > 1)
                right = Sort(right);
            //сортировка слиянием половинок
            buff = MergeSort(left, right);
        }
        //возврат отсортированного массива
        return buff;
    }
    static int[] MergeSort(int[] left, int[] right)
    {
        //буфер для отсортированного массива
        int[] buff = new int[left.Length + right.Length];
        //счетчики длины трех массивов
        int i = 0;  //соединенный массив
        int l = 0;  //левый массив
        int r = 0;  //правый массив
        //сортировка сравнением элементов
        for (; i < buff.Length; i++)
        {
```

```
            //если правая часть уже использована, дальнейшее движе
               ние происходит только в левой
            //проверка на выход правого массива за пределы
            if (r >= right.Length)
            {
                buff[i] = left[l];
                l++;
            }
            //проверка на выход за пределы левого массива
            //и сравнение текущих значений обоих массивов
            else if (l < left.Length && left[l] < right[r])
            {
                buff[i] = left[l];
                l++;
            }
            //если текущее значение правой части больше
            else
            {
                buff[i] = right[r];
                r++;
                //подсчет количества инверсий
                if (l < left.Length)
                    number += left.Length - l;
            }
        }
        //возврат отсортированного массива
        return buff;
    }

    public int[] FetchData(Task task)
    {
        Log.Print("Client has fetched data");
        int[] temp = new int[task.stop-task.start];
        int j = 0;
        for (int i = task.start; i < task.stop; i++)
        {
            temp[j] = dataArray[i];
            j++;
        }

        return temp;
```

```
    }

    public Task GetTask()
    {
        Log.Print("Client has requested task");
        lock (tasksLock)
        {
            if (pendingTasks.Count == 0)
            {
                Log.Print("No more tasks");
                return null;
            }
            else
                return pendingTasks.Dequeue();
            //return (pendingTasks.Count == 0 ? null :
              pendingTasks.Dequeue());
        }
    }

    public void Finish(Task task, int[] data)
    {
        Log.Print("Client has finished task");
        lock (dataLock)
        {
            int j = 0;
            for (int i = task.start; i < task.stop; i++)
            {
                dataArray[i] = data[j];
                j++;
                Console.Out.Write(dataArray[i]+"  ");
            }
            Console.Out.WriteLine();
        }
        //finishedTasks.Add(task);
        if (pendingTasks.Count == 0)
        {
            Log.Print("Final task has finished");
            Console.Out.WriteLine("\n\n");
            Console.Out.WriteLine("Sorting an array(Client)");
            for (int i = 0; i < 100; i++)
            {
```

```
                    Console.Out.Write(dataArray[i] + " ");
                }
                Console.Out.WriteLine("\n\n");
                dataArray = Sort(dataArray);

                Console.Out.WriteLine("Sorting an array(MERGE on
                    server)");
                for (int i = 0; i < 100; i++)
                {
                    Console.Out.Write( dataArray[i]+" ");
                }
                Console.Out.WriteLine();
            }
        }
    }
    [Serializable]
    public class Task
    {
        public int start = 0;
        public int stop = 0;
        //public List<int> indexes;
        //public Task()
        //{
        //    indexes = new List<int>();
        //}
    }

    public class Log
    {
        // вывести время и msg
        public static void Print(String msg)
        {
            System.Console.WriteLine("[" + DateTime.Now.Hour.ToString
                () + ":" +
                DateTime.Now.Minute.ToString() + ":" + DateTime.Now.
                    Second.ToString()
            + "] " + msg);
        }
    }
}
```

2.3 Результат работы программы

Скриншот работы первого клиента представлен на Рис.1.

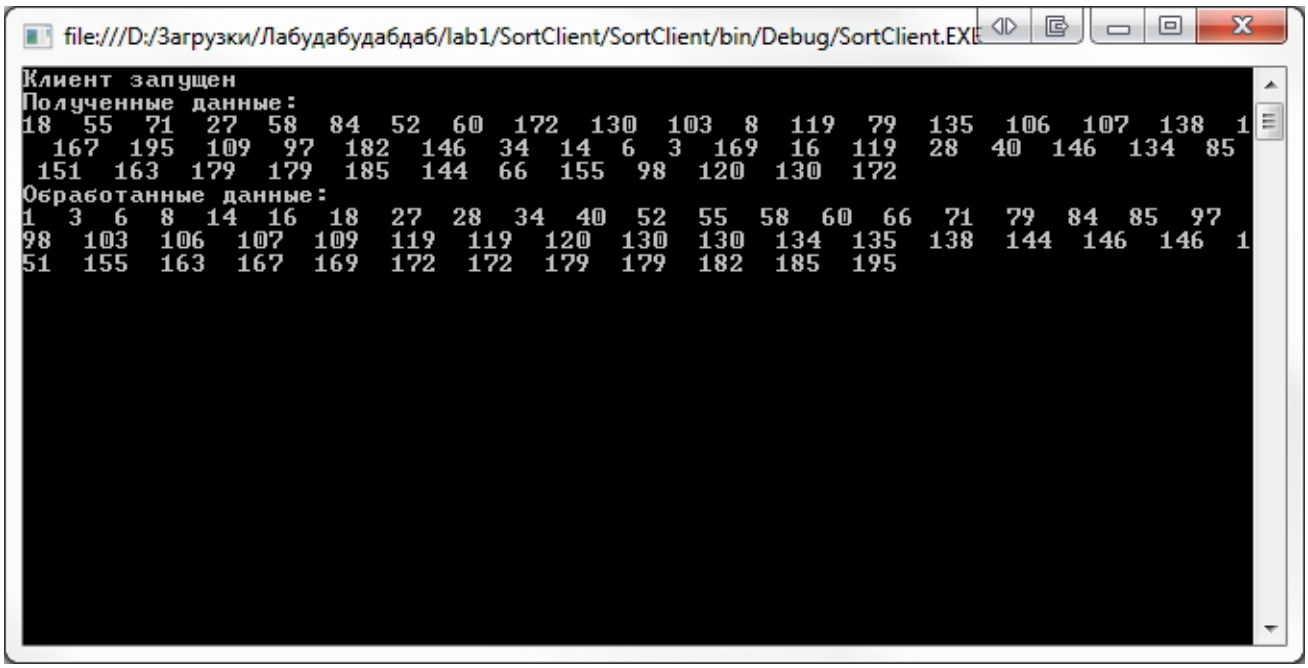

Рисунок 1

Скриншот работы второго клиента представлен на Рис.2.

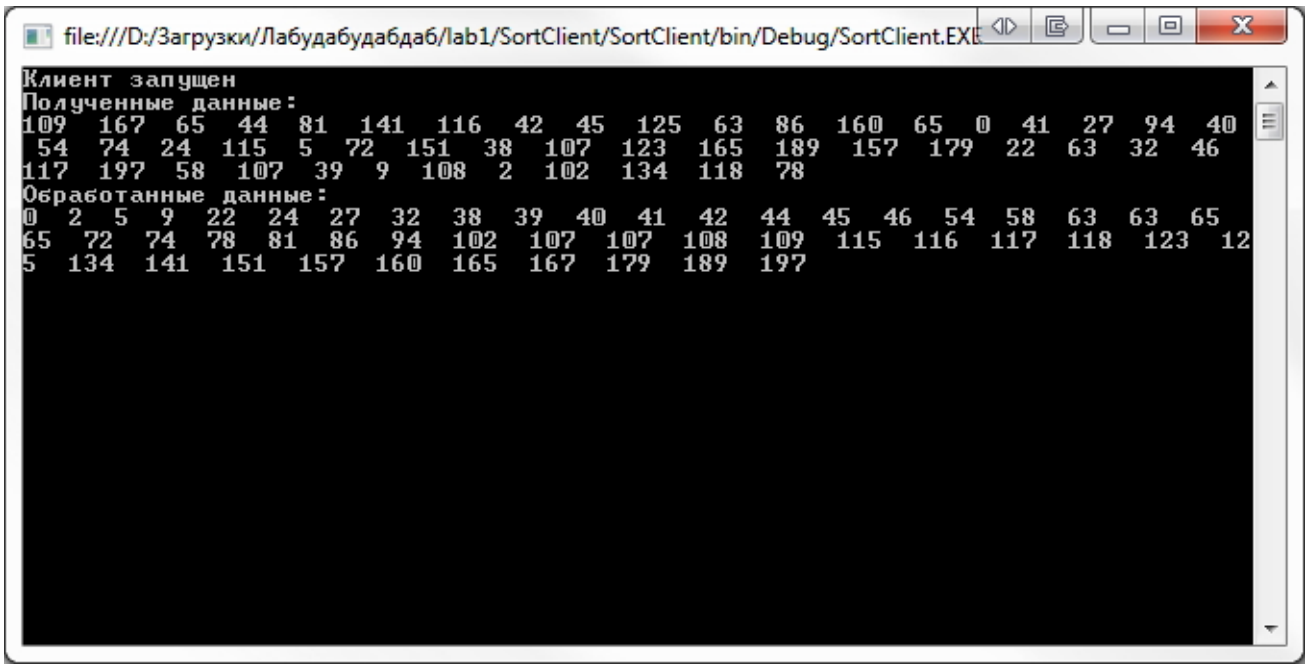

Рисунок 2

Скриншот работы сервера представлен на Рис.3.

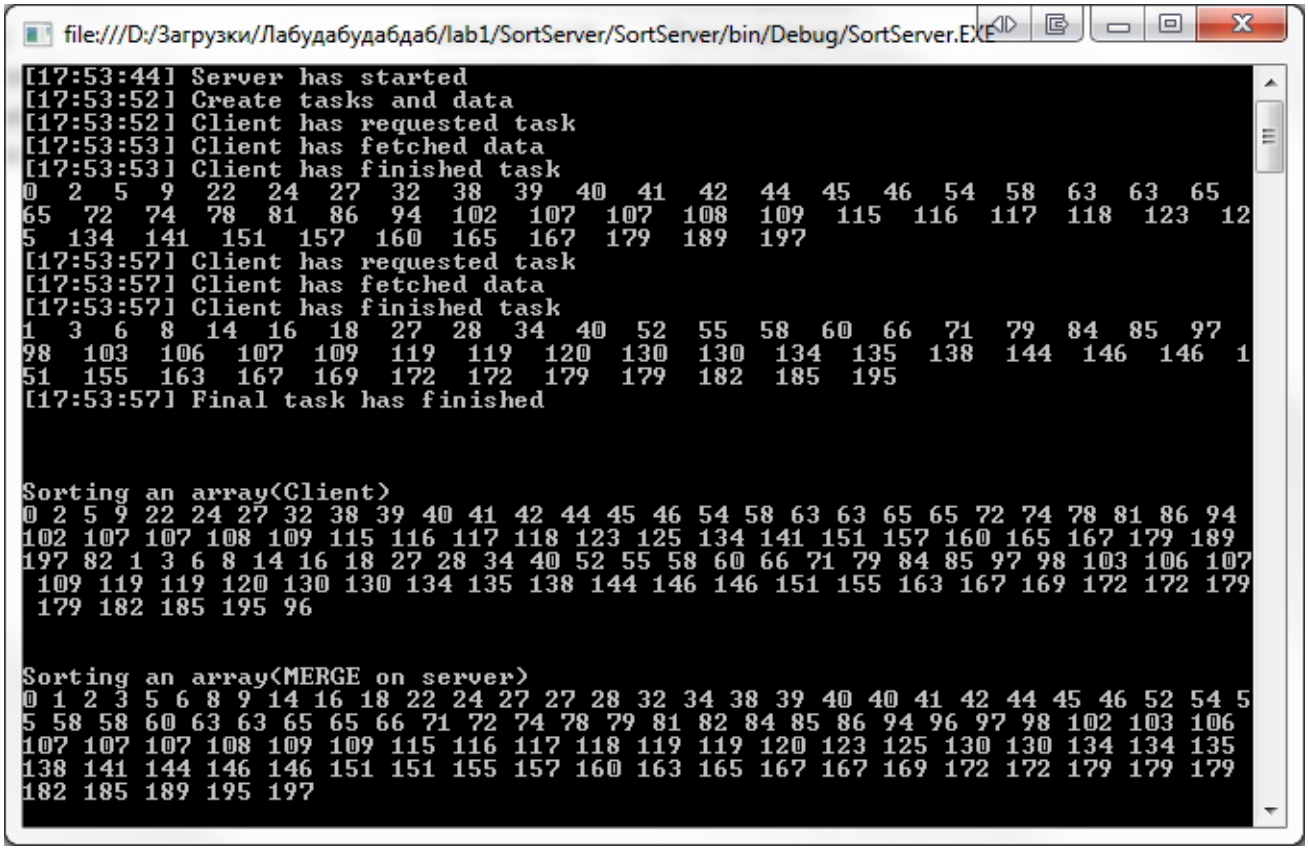

Рисунок 3

3 ВЫВОД

В результате выполнения лабораторной работы был получен программный комплекс, состоящий из сервера и клиента и реализующий алгоритм поиска минимального и максимального элементов массива.