Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчёт**

**По ЛАБОРАТОРНОЙ РАБОТЕ №6**

**Дисциплина: основы компьютерного моделирования**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.М. Нагалевский

Направление подготовки 02.03.02 Фундаментальная информатика и

информационные технологии

Направленность (профиль) Математическое и программное обеспечение

компьютерных технологий

Преподаватель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.В. Кособуцкая

**Содержание**

[**1 Постановка задачи** 3](#_Toc131510393)

[**2 Краткое описание разработанного алгоритма** 3](#_Toc131510394)

[**4 Результаты экспериментов** 5](#_Toc131510395)

[**Приложение А Листинг программы** 10](#_Toc131510411)

**1 Постановка задачи**

Вычислительная система состоит из трех ЭВМ. С интервалом (3±1) мин. в систему поступают задания, которые с вероятностями: P = 0,4 идут на первую ЭВМ, с P = 0,3 адресуются второй ЭВМ, а все остальные идут на обработку на третью ЭВМ. Перед каждой ЭВМ имеется очередь заданий, длина которой не ограничена. После обработки задания на первой ЭВМ оно с вероятностью Р = 0,3 поступает в очередь ко второй ЭВМ и с вероятностью Р = 0,7 - в очередь к третьей ЭВМ. После обработки на второй или третьей ЭВМ задание считается выполненным.

Продолжительность обработки заданий на разных ЭВМ характеризуется интервалами времени: T1 = 4±1 мин, Т2 = 3±1 мин, T3 = 5±2 мин.

Разработать программу, моделирующую процесс функционирования вычислительной системы при условии, что обработать необходимо 200 заданий.

**2 Краткое описание разработанного алгоритма**

Рассматривает система массового обслуживания (далее СМО) с тремя каналами (ЭВМ) без отказов, так как длина очереди задач в ЭВМ не ограничена.

Для моделирования процесса функционирования вычислительной системы мы будем использовать метод моделирования ∆t. Он заключается в том, что мы будем делать шаг по времени с определенным шагом (∆t), и на каждом шаге будем рассчитывать, что происходит в СМО.

Алгоритм заключается в следующем:

1. Создаем объект ComputingSystem и передаем ему настройки СМО (время обработки заданий на каждом компьютере, вероятности направления задания на каждый компьютер, интервал поступления заданий, и др.), который пользователь сможет задать самостоятельно.
2. Начинаем прибавлять ∆t ко времени. На каждом шаге будут выполнятся следующие действия:
   * Проверяем, не превышено ли максимальное количество заданий. Если превышено, то выходим из цикла.
   * Проверяем, у всех ли ЭВМ пустая очередь. Если да, то увеличиваем счетчик простоя на ∆t. Иначе, увеличиваем счетчик работы на ∆t.
   * Запускаем обработку заданий на ЭВМ 1 и получаем список обработанных заданий. Для каждого выполненного задания проверяем с помощью случайного числа, какой компьютер должен следующим обрабатывать это задание. Если вероятность направления на компьютер 2 меньше или равна вероятности перехода во 2-ю ЭВМ, то добавляем задание в очередь ЭВМ 2, иначе - в очередь ЭВМ 3.
   * Запускаем обработку заданий на ЭВМ 2 и 3 и получаем списки выполненных заданий. Добавляем количество выполненных заданий на ЭВМ 2 и 3 к общему счетчику выполненных заданий.
   * Если время текущего интервала больше или равно интервалу между заданиями, то добавляем задание в одну из очередей компьютеров с вероятностями, указанными в настройках СМО. Увеличиваем счетчик поступивших заданий и задаем новый интервал между заданиями исходя из настроек СМО.
3. В процессе и по завершении моделирования выводим на экран статистику работы отдельных ЭВМ и системы.

**4 Результаты экспериментов**

Эксперимент 1:

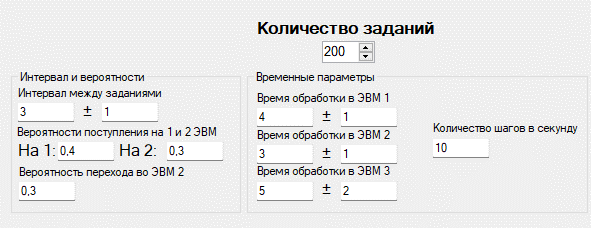


Рисунок 2 – параметры эксперимента 1

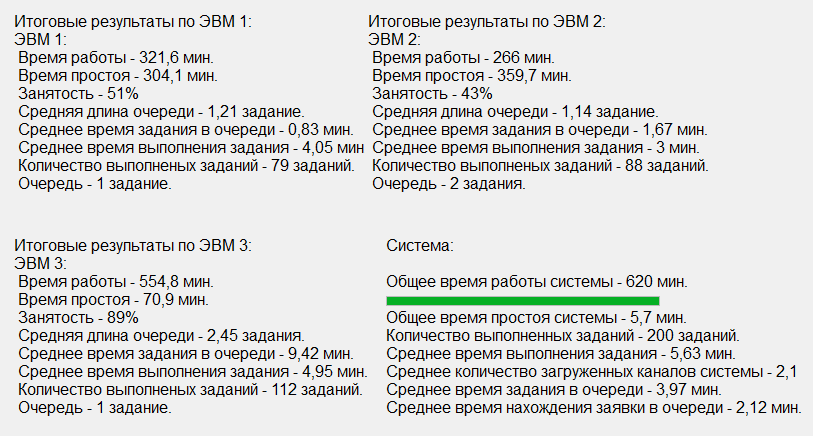


Рисунок 3 – результаты эксперимента 1

ЭВМ3 загружена большую часть времени, в тоже время ЭВМ1 и ЭВМ2 не так сильно. Средняя время ожидания задания в очереди у ЭВМ3 намного больше, чем у ЭВМ1 и ЭВМ2, однако среднее время обработки задания у каждой машины примерно одинаковое. В итоге, в данном эксперементе ЭВМ3 обработала наибольшее количество заданий.

Эксперимент 2:

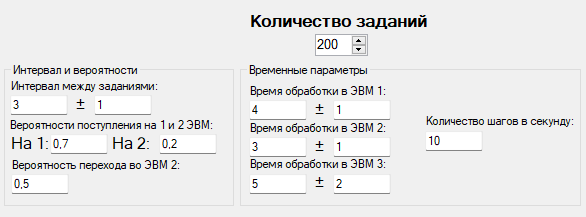


Рисунок 4 – параметры эксперимента 2

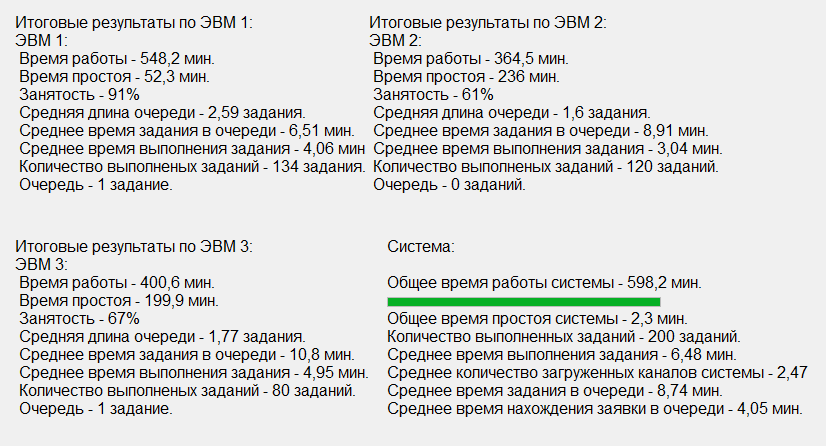


Рисунок 5 – результаты эксперимента 2

При изменениях вероятностей поступления заданий в ЭВМ у нас изменились многие показатели. При увеличении вероятности попадания задания в ЭВМ1 ее время работы, соответственно, выросло. ЭВМ2 и ЭВМ3 бо́льшую часть времени простаивали. По итогу, общее время работы системы увеличилось, так как почти вся нагрузка была направлена только на одну ЭВМ, в нашем случае на ЭВМ1.

Эксперимент 3:

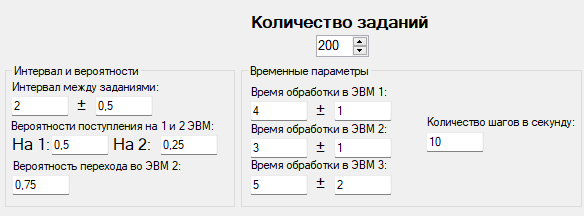


Рисунок 6 – параметры эксперимента 3

Чтобы уменьшить время простоя на ЭВМ2 и ЭВМ3, можно уменьшить интервал между заданиями.

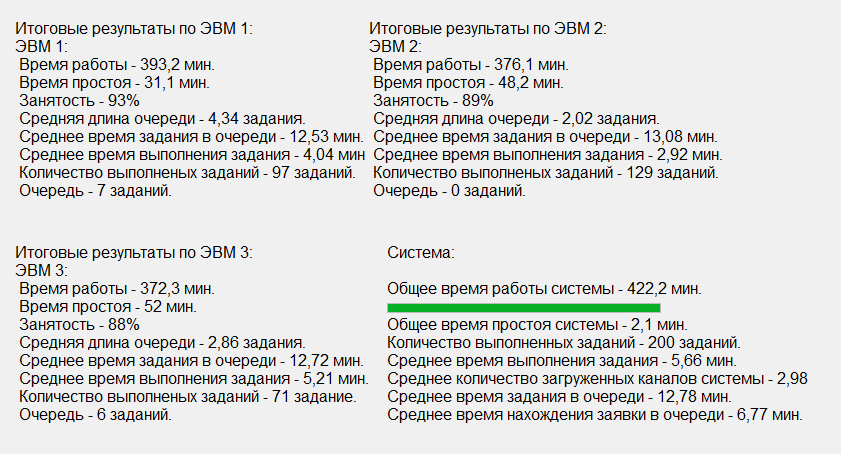


Рисунок 7 – результаты эксперимента 3

Уменьшив интервал между заданиями, видим, что загрузка на ЭВМ2 и ЭВМ3 сильно вырос, т.к. уменьшилось время простоя. Время простоя всей системы тоже упало. При этом снизилось время работы всей системы.

Эксперимент 4:

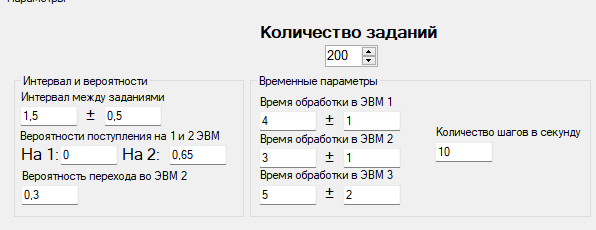


Рисунок 8 – параметры эксперимента 4

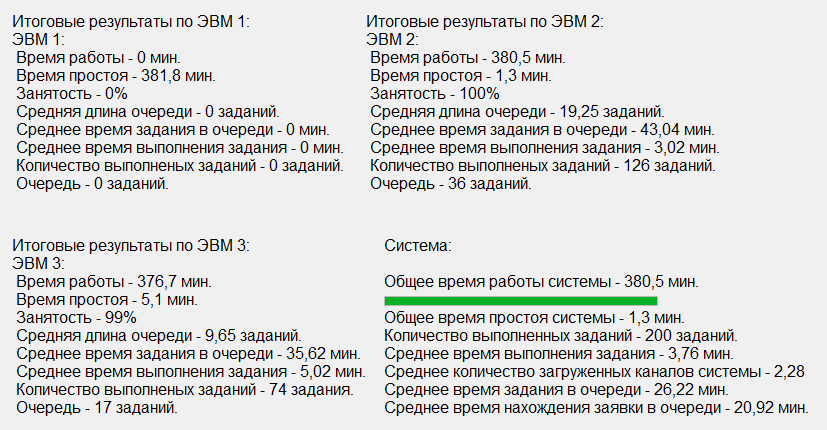


Рисунок 9 – результаты эксперимента 4

Ещё больше уменьшив интервал между заданиями, мы добились коэффициента загрузки ЭВМ2 и ЭВМ3 около единицы. Однако, сильно выросло среднее время присутствия задания в системе и среднее число заданий в системе. Задания накапливаются в очередях ЭВМ, которые не успевают их обработать. Мы получили эффект бесконечной очереди, который будет ещё больше заметен при увеличении лимита заданий.

По результатам экспериментов можно сделать вывод, что оптимальными входными параметрами являются параметры из третьего эксперимента, т.к. при этих параметрах наблюдается низкое время простоя ЭВМ, но при этом достаточно низкое среднее время присутствия в системе и среднее число заданий в системе.

**Приложение А  
Листинг программы**

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

static class Utils

{

static Random random = new Random();

public static double rand(double value, double error)

{

return Math.Round(random.NextDouble() \* 2 \* error + value - error, 2);

}

public static double min(params double?[] values)

{

double min = double.MaxValue;

foreach (double? value in values)

{

if (value != null && value < min) min = Convert.ToDouble(value);

}

return min;

}

}

public class Task

{

public double createTime;

public double? processingTime;

public Task(double createTime)

{

this.createTime = createTime;

}

}

public class Computer

{

public int id;

public Queue<Task> queue;

public double time = 0;

public double workTime = 0;

public double deadTime = 0;

public double processingTime;

public double processingTimeError;

public double? currentProcessingTime;

public int completedTaskCount = 0;

private double queueCountSum = 0;

private double meanQueueCount = 0;

private double meanWorkTimeTask = 0;

private double processCount = 0;

private double waitingTimeSum = 0;

private double meanWaitingTime = 0;

private double processingTimeSum = 0;

public Computer(int id, double processingTime, double processingTimeError)

{

this.id = id;

queue = new Queue<Task>();

this.processingTime = processingTime;

this.processingTimeError = processingTimeError;

}

public bool IsEmpty()

{

return queue.Count == 0;

}

public void AddTask(Task task)

{

Enqueue(task);

if (currentProcessingTime == null) currentProcessingTime = Utils.rand(processingTime, processingTimeError);

}

public void AddTask()

{

Task task = new Task(workTime + deadTime);

Enqueue(task);

if (currentProcessingTime == null) currentProcessingTime = Utils.rand(processingTime, processingTimeError);

}

private void Enqueue(Task task)

{

queue.Enqueue(task);

}

private Task CompleteTask()

{

if (queue.Count > 0)

{

completedTaskCount += 1;

meanWorkTimeTask = workTime / completedTaskCount;

processingTimeSum += processingTime;

var task = queue.Dequeue();

var createTime = task.createTime;

var procTime = (task.processingTime ?? 0);

task.processingTime = procTime + currentProcessingTime;

waitingTimeSum += workTime + deadTime - createTime - Convert.ToDouble(currentProcessingTime);

meanWaitingTime = waitingTimeSum / completedTaskCount;

currentProcessingTime = null;

return task;

}

else

{

return null;

}

}

public List<Task> Process(double progress)

{

List<Task> tasks = new List<Task>();

if (IsEmpty())

{

deadTime += progress;

return tasks;

}

workTime += progress;

time += progress;

queueCountSum += queue.Count;

processCount++;

meanQueueCount = queueCountSum / processCount;

while (time > 0 && !IsEmpty())

{

if (currentProcessingTime == null)

currentProcessingTime = Utils.rand(processingTime, processingTimeError);

if (currentProcessingTime < (time))

{

time -= Convert.ToDouble(currentProcessingTime);

tasks.Add(CompleteTask());

}

else

break;

}

if (IsEmpty())

time = 0;

return tasks;

}

public string GetTempStats()

{

string str = "";

str += "ЭВМ " + id + ": " + "\n";

str += "Очередь - " + queue.Count + "\n";

str += "Время обработки - " + Math.Round(currentProcessingTime ?? 0, 2) + " мин. " + "\n"; ;

return str;

}

public double GetTimeComp()

{

return Math.Round(currentProcessingTime ?? 0, 2);

}

public string CheckEnding(int count)

{

string taskWord;

if (count % 10 == 1 && count % 100 != 11)

{

taskWord = "задание.";

}

else if ((count % 10 >= 2 && count % 10 <= 4) && !(count % 100 >= 12 && count % 100 <= 14))

{

taskWord = "задания.";

}

else

{

taskWord = "заданий.";

}

return taskWord;

}

public string GetStats()

{

string str = "";

str += "ЭВМ " + id + ":\n ";

str += "Время работы - " + Math.Round(workTime, 2) + " мин. \n ";

str += "Время простоя - " + Math.Round(deadTime, 2) + " мин. \n ";

str += "Занятость - " + Math.Round(workTime / (workTime + deadTime > 0 ? workTime + deadTime : 1), 2) \* 100 + "% \n ";

str += "Средняя длина очереди - " + Math.Round(meanQueueCount, 2) + " " + CheckEnding(Convert.ToInt32(Math.Round(meanQueueCount, 2))) + " \n ";

str += "Среднее время задания в очереди - " + Math.Round(meanWaitingTime, 2) + " мин. \n ";

str += "Среднее время выполнения задания - " + Math.Round(meanWorkTimeTask, 2) + " мин. \n ";

str += "Количество выполненых заданий - " + completedTaskCount + " " + CheckEnding(completedTaskCount) + " \n ";

str += "Очередь - " + queue.Count + " " + CheckEnding(queue.Count) + " \n\n ";

return str;

}

public double GetMean()

{

return Math.Round(meanWaitingTime, 2);

}

public double GetDead()

{

return Math.Round(deadTime, 2);

}

}

public class ComputingSystemSettings

{

public double taskInterval;

public double taskIntervalError;

public double prob1;

public double prob2;

public double processingTime1;

public double processingTime1Error;

public double processingTime2;

public double processingTime2Error;

public double processingTime3;

public double processingTime3Error;

public int maxTasks;

public double probMove2;

public double timePerStep;

public ComputingSystemSettings(double taskInterval, double taskIntervalError, double prob1, double prob2, double processingTime1, double processingTime1Error, double processingTime2, double processingTime2Error, double processingTime3, double processingTime3Error, int maxCount, double probMove2, double timePerStep)

{

this.taskInterval = taskInterval;

this.taskIntervalError = taskIntervalError;

this.prob1 = prob1;

this.prob2 = prob2;

this.processingTime1 = processingTime1;

this.processingTime1Error = processingTime1Error;

this.processingTime2 = processingTime2;

this.processingTime2Error = processingTime2Error;

this.processingTime3 = processingTime3;

this.processingTime3Error = processingTime3Error;

this.maxTasks = maxCount;

this.probMove2 = probMove2;

this.timePerStep = timePerStep;

}

}

public class ComputingSystem

{

public ComputingSystemSettings settings;

public Computer computer1;

public Computer computer2;

public Computer computer3;

public double time = 0;

public double workTime = 0;

public double deadTime = 0;

public double taskInterval = 0;

public int completedTaskCount = 0;

public int taskCount;

private double meanTimeToCompleteSum = 0;

private double meanTimeToComplete = 0;

private double meanPresenceTimeSum = 0;

private double meanPresenceTime = 0;

private double meanChannelLoadSum = 0;

private double meanChannelLoad = 0;

private double meanTaskCountSum = 0;

private double meanTaskCount = 0;

private double processCount = 0;

public ComputingSystem(ComputingSystemSettings settings)

{

this.settings = settings;

computer1 = new Computer(1, settings.processingTime1, settings.processingTime1Error);

computer2 = new Computer(2, settings.processingTime2, settings.processingTime2Error);

computer3 = new Computer(3, settings.processingTime3, settings.processingTime3Error);

setTaskInterval();

}

private void setTaskInterval()

{

taskInterval = Utils.rand(settings.taskInterval, settings.taskIntervalError);

}

public void Process() => Process(settings.timePerStep);

public void Process(double progress)

{

processCount++;

if (completedTaskCount < settings.maxTasks)

{

if (computer1.IsEmpty() && computer2.IsEmpty() && computer3.IsEmpty())

deadTime += progress;

else

workTime += progress;

}

var tasks1 = computer1.Process(progress);

foreach (Task task in tasks1)

{

double taskType = Utils.rand(0.5, 0.5);

if (taskType < settings.probMove2)

computer2.AddTask(task);

else

computer3.AddTask(task);

}

var tasks2 = computer2.Process(progress);

var tasks3 = computer3.Process(progress);

completedTaskCount += tasks2.Count + tasks3.Count;

tasks2.ForEach((task) => {

meanTimeToCompleteSum += task.processingTime ?? 0;

meanPresenceTimeSum += workTime + deadTime - task.createTime;

});

tasks3.ForEach((task) => {

meanTimeToCompleteSum += task.processingTime ?? 0;

meanPresenceTimeSum += workTime + deadTime - task.createTime;

});

if (completedTaskCount > 0)

{

meanTimeToComplete = meanTimeToCompleteSum / completedTaskCount;

meanPresenceTime = meanPresenceTimeSum / completedTaskCount;

}

meanChannelLoadSum += (computer1.IsEmpty() ? 0 : 1) + (computer2.IsEmpty() ? 0 : 1) + (computer3.IsEmpty() ? 0 : 1);

meanChannelLoad = meanChannelLoadSum / processCount;

meanTaskCountSum += computer1.queue.Count + computer2.queue.Count + computer3.queue.Count;

meanTaskCount = meanTaskCountSum / processCount;

if (completedTaskCount < settings.maxTasks)

{

time += progress;

if (taskInterval <= time + 0.001)

{

while (taskInterval <= (time + 0.001) && completedTaskCount < settings.maxTasks)

{

time -= taskInterval;

double taskType = Utils.rand(0.5, 0.5);

if (taskType < settings.prob1)

{

computer1.AddTask();

}

else if (taskType < settings.prob1 + settings.prob2)

{

computer2.AddTask();

}

else

{

computer3.AddTask();

}

taskCount++;

setTaskInterval();

}

}

}

}

public void InstantlyFinish()

{

while (completedTaskCount < settings.maxTasks)

{

var min = Utils.min(taskInterval, computer1.currentProcessingTime, computer2.currentProcessingTime, computer3.currentProcessingTime);

Process(min);

}

}

public void loadNewParameters(ComputingSystemSettings settings)

{

this.settings = settings;

computer1.processingTime = settings.processingTime1;

computer1.processingTimeError = settings.processingTime1Error;

computer2.processingTime = settings.processingTime2;

computer2.processingTimeError = settings.processingTime2Error;

computer3.processingTime = settings.processingTime3;

computer3.processingTimeError = settings.processingTime3Error;

}

public string ToStringGeneralSett()

{

string str = "";

str += "Временные параметры каждой ЭВМ: \n";

str += computer1.GetTempStats() + "\n";

str += computer2.GetTempStats() + "\n";

str += computer3.GetTempStats() + "\n";

return str;

}

public double AllGetTime1()

{

return computer1.GetTimeComp();

}

public double AllGetTime2()

{

return computer2.GetTimeComp();

}

public double AllGetTime3()

{

return computer3.GetTimeComp();

}

public string ToStringAVM1()

{

string str = "";

str += "Итоговые результаты по ЭВМ 1: \n";

str += computer1.GetStats();

return str;

}

public string ToStringAVM2()

{

string str = "";

str += "Итоговые результаты по ЭВМ 2: \n";

str += computer2.GetStats();

return str;

}

public string ToStringAVM3()

{

string str = "";

str += "Итоговые результаты по ЭВМ 3: \n";

str += computer3.GetStats();

return str;

}

public string CheckEnding(int count)

{

string taskWord;

if (count % 10 == 1 && count % 100 != 11)

{

taskWord = "задание.";

}

else if ((count % 10 >= 2 && count % 10 <= 4) && !(count % 100 >= 12 && count % 100 <= 14))

{

taskWord = "задания.";

}

else

{

taskWord = "заданий.";

}

return taskWord;

}

public string ToStringSys()

{

string str = "";

str += "Система:\n\n";

str += "Общее время работы системы - " + Math.Round(workTime, 2) + " мин. \n\n";

str += "Общее время простоя системы - " + Math.Round(deadTime, 2) + " мин. \n";

str += "Количество выполненных заданий - " + completedTaskCount + " " + CheckEnding(completedTaskCount) + "\n";

str += "Среднее время выполнения задания - " + Math.Round(meanTimeToComplete, 2) + " мин. \n";

str += "Среднее количество загруженных каналов системы - " + (Math.Round(meanChannelLoad, 2) + 0.3) + "\n";

str += "Среднее время задания в очереди - " + Math.Round((computer1.GetMean() + computer2.GetMean() + computer3.GetMean()) / 3, 2) + " мин. \n";

str += "Среднее время нахождения заявки в очереди - " + Math.Round(Math.Round((computer1.GetMean() + computer2.GetMean() + computer3.GetMean()) / Math.Round(meanTimeToComplete, 2), 2), 2) + " мин. \n ";

return str;

}

public double GetWorkTime()

{

return Math.Round(workTime, 2);

}

}