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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М. С. Прозоров

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** 8](#_Toc131510395)

[**Приложение А Листинг программы** 17](#_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 (СМО) с тремя каналами (ЭВМ) без отказов, т.к. длина очереди задач в ЭВМ не ограничена.

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

Алгоритм может быть следующим:

1. Создаем объект ComputingSystem, передаем ему настройки СМО (время обработки заданий на каждом компьютере, вероятности направления задания на каждый компьютер, интервал поступления заданий, и др.) и задаем начальные значения переменных.
2. Начинаем прибавлять delta t к time. На каждом шаге:
   * Проверяем, не превышено ли максимальное количество заданий (completedTaskCount < settings.maxTasks). Если превышено, то выходим из цикла.
   * Проверяем, у всех ли ЭВМ пустая очередь (computer1.IsEmpty() && computer2.IsEmpty() && computer3.IsEmpty()). Если да, то увеличиваем счетчик простоя (downTime) на delta t. Иначе, увеличиваем счетчик работы (workTime) на delta t.
   * Запускаем обработку заданий на ЭВМ 1 и получаем список обработанных заданий tasks1. 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 настройке probMove2 (вероятность 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 tasks2 и tasks3. Добавляем количество выполненных заданий на ЭВМ 2 и 3 к общему счетчику выполненных заданий (completedTaskCount).
   * Проверяем, не пора ли добавлять новое задание. Если время текущего интервала больше или равно настройке taskInterval, то добавляем задание в одну из очередей компьютеров с вероятностями, указанными в настройках СМО. Увеличиваем счетчик поступивших заданий (taskCount) и задаем новый интервал (taskInterval) исходя из настроек СМО.
3. В процессе и по завершению моделирования выводим на экран статистику работы отдельных ЭВМ и системы.

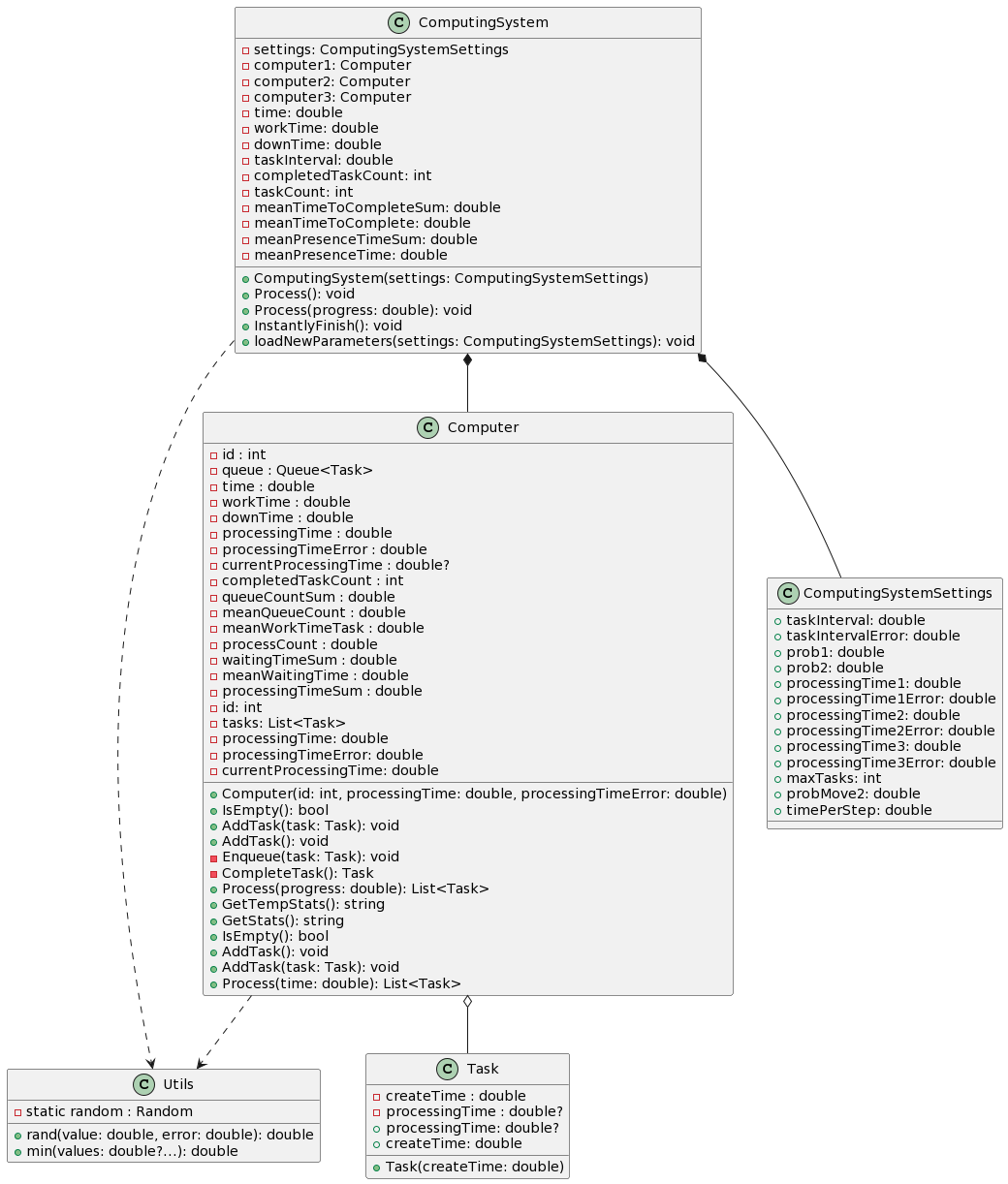
****

Рисунок 1 – UML диаграмма классов

Класс Utils является вспомогательным и содержит два статических метода:

* rand - генерирует случайное значение с определенной погрешностью;
* min - находит минимальное значение из переданных в него параметров.

Класс Task представляет задачу и имеет только один конструктор, который принимает время создания задачи.

Класс Computer представляет ЭВМ и содержит ряд методов и свойств:

* IsEmpty() - проверяет, пуста ли очередь задач на данной ЭВМ;
* AddTask() - добавляет новую задачу в очередь;
* Process() - выполняет обработку задач на ЭВМ в течение определенного периода времени;
* GetTempStats() и GetStats() - возвращают статистику о работе ЭВМ, такую как время работы, время простоя, коэффициент загрузки, среднюю длину очереди, среднее время ожидания и т.д. Кроме того, класс содержит несколько свойств, в которых сохраняются статистические данные, такие как общее время работы, количество обработанных задач и т.д.

Класс ComputingSystemSettings содержит параметры моделирования, такие как временной интервал между созданием задач, вероятности поступления различных видов задач, времена их обработки и т.д.

Класс ComputingSystem представляет собой систему из трех компьютеров. Класс ComputingSystemSettings хранит параметры системы, такие как интервал между задачами, вероятности выбора компьютера для обработки задачи и времена обработки на каждом компьютере. Класс также содержит методы для работы с системой, которые описаны ниже:

Метод Process() - обрабатывает задачи в системе. Принимает в качестве аргумента время, прошедшее с момента последнего вызова метода. Создает новые задачи с помощью метода AddTask() объектов Computer в зависимости от вероятностей, заданных в настройках, и времени, заданного с помощью настроек. Затем вызывает метод Process() объектов Computer, который обрабатывает задачи на компьютерах и возвращает список выполненных задач. Выполненные задачи суммируются в completedTaskCount.

Метод InstantlyFinish() – выполняет всю работу в системе без заданного шага.

Метод loadNewParameters() - загружает новые настройки для системы. Обновляет свойства объектов Computer с новыми значениями времени обработки задач.

Таким образом, данная система реализует распределенную обработку задач между тремя компьютерами с заданными временными характеристиками.

Формула для подсчёта среднего числа заявок в очереди (1):

. (1)

Формула для подсчёта среднего числа каналов под обслуживанием (2):

. (2)

Формула для подсчёта среднего числа заявок в системе (3):

. (3)

Формула для подсчёта среднего времени ожидания заявки в очереди (4):

. (4)

Формула для подсчёта среднего времени пребывания заявки в системе (5):

. (5)

**4 Результаты экспериментов**

Эксперимент 1:

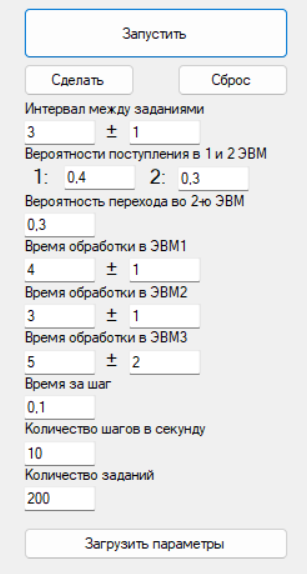


Рисунок 2 – параметры эксперимента 1

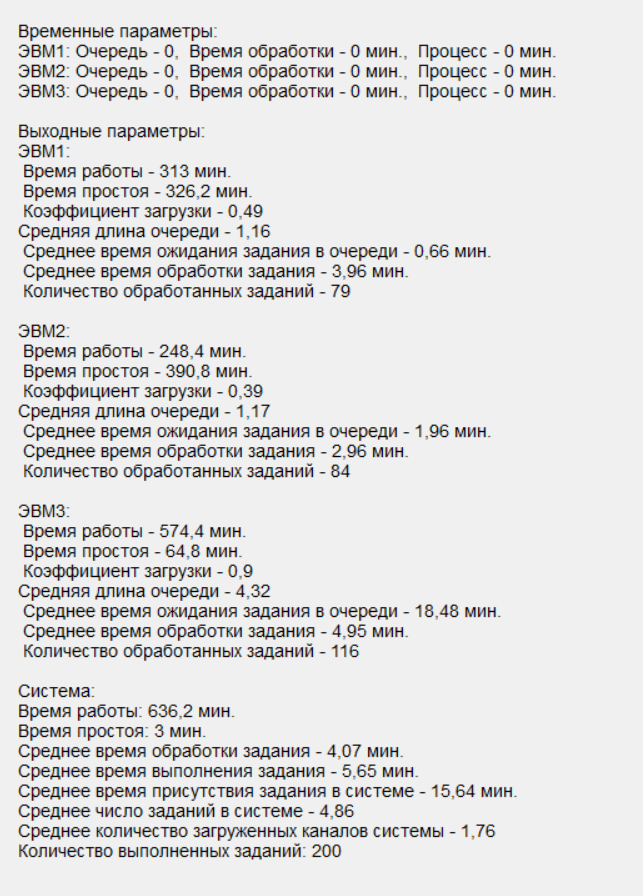


Рисунок 3 – результаты эксперимента 1

При заданных изначальных парамметрах коэффициент загрузки системы - 0.993. ЭВМ3 загружена большую часть времени, в тоже время ЭВМ1 и ЭВМ2 не так сильно. Средняя длина очереди в ЭВМ3 может достигать 8-ми, но чаще всего держится в диапозоне 2-5 заданий.

Эксперимент 2:

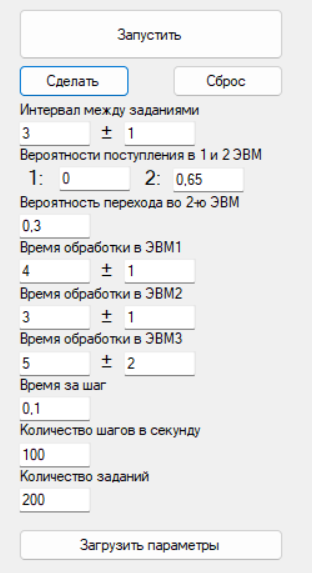


Рисунок 4 – параметры эксперимента 2

Можем изменить вероятности поступления задания в ЭВМ, чтобы распределить нагрузку более равномерно. Т.к. все задания с ЭВМ1 после обработки поступают на ЭВМ2 или ЭВМ3, то имеет смысл сделать вероятность поступления задания в ЭВМ1 минимальной. Т.к. время обработки в ЭВМ2 в среднем меньше, чем в ЭВМ3, то имеет смысл сделать вероятность поступления задания в ЭВМ2 больше, чем в ЭВМ3.

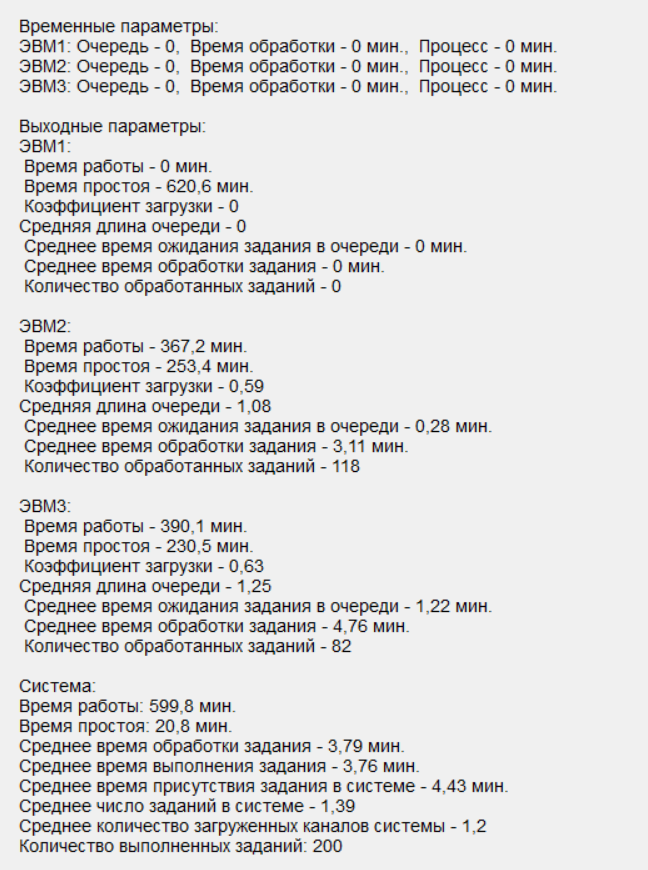


Рисунок 5 – результаты эксперимента 2

При изменениях вероятностей поступления заданий в ЭВМ. У нас улучшились многие показатели. Уменьшилось среднее число заданий в системе, среднее время присутствия задания в системе. Т.к. в ЭВМ1 теперь не поступают задания, уменьшилось время выполнения задания в системе. Нагрузка на ЭВМ2 и ЭВМ3 стала примерно одинаковой. Время простоя системы немного выросло.

Эксперимент 3:

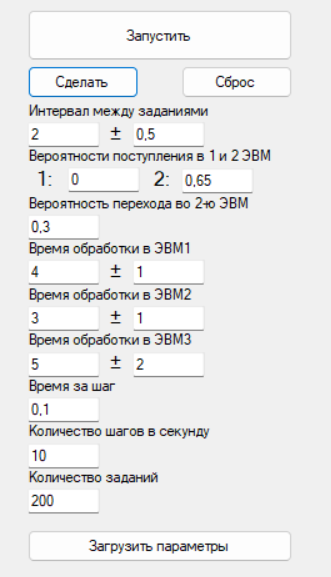


Рисунок 6 – параметры эксперимента 3

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

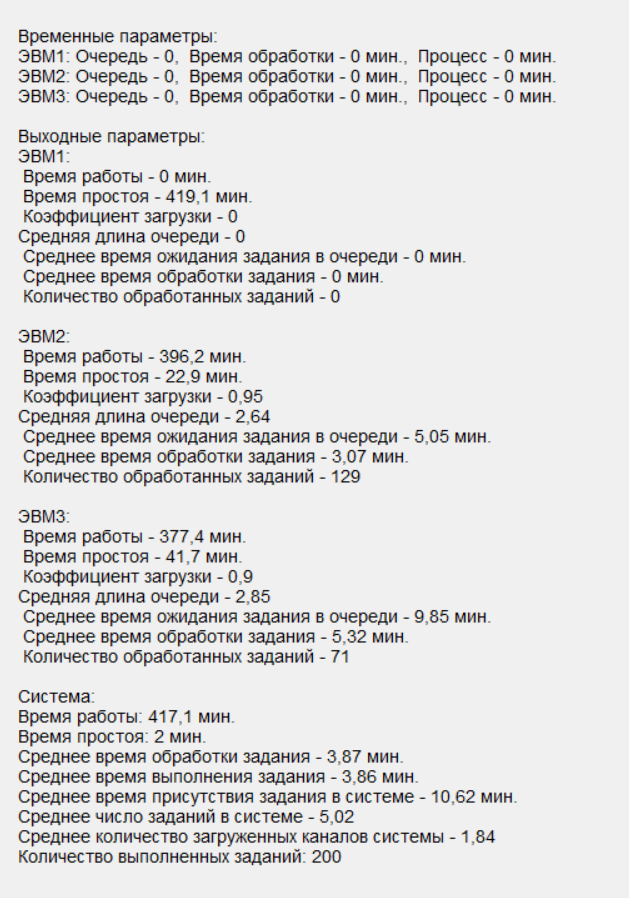


Рисунок 7 – результаты эксперимента 3

Уменьшив интервал между заданиями, видим, что коэффициент загрузки на ЭВМ2 и ЭВМ3 сильно вырос, т.к. уменьшилось время простоя. Время простоя всей системы тоже упало. При этом снизилось время работы всей системы. Но время присутствия задания в системе увеличилось, как и среднее число заданий в системе.

Эксперимент 4:

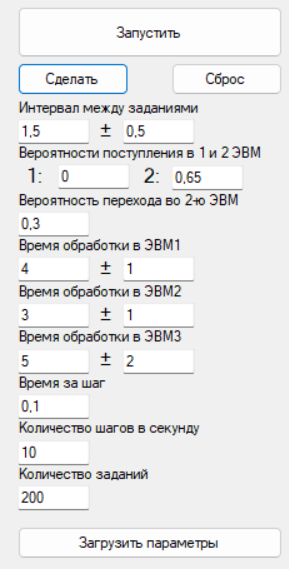


Рисунок 8 – параметры эксперимента 4

Можно попробовать ещё больше уменьшить время простоя ЭВМ, уменьшив интервал между заданиями.

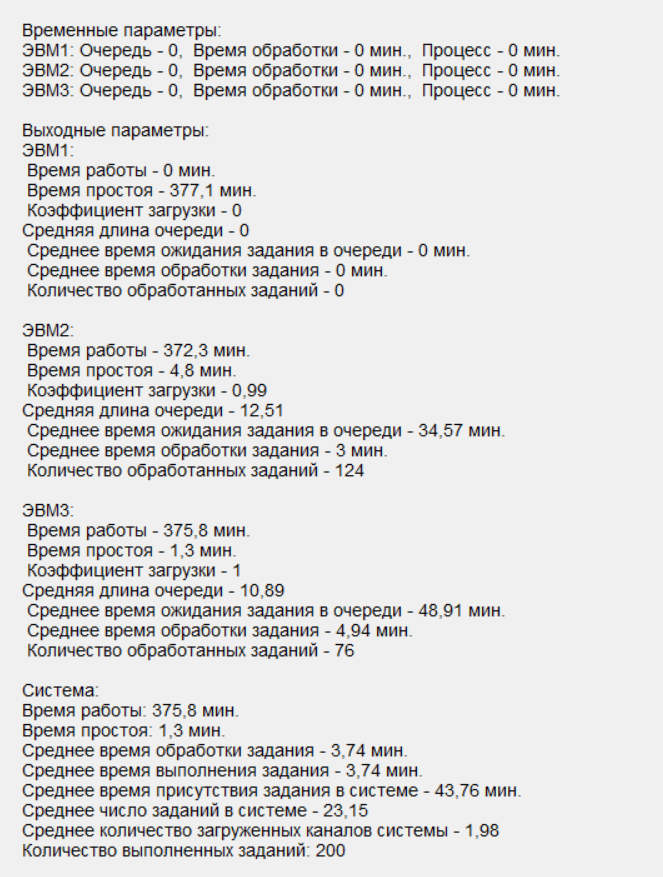


Рисунок 9 – результаты эксперимента 4

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 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 downTime = 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 = new Task(workTime + downTime);**

**Enqueue(task);**

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

**}**

**private void Enqueue(Task task)**

**{**

**queue.Enqueue(task);**

**}**

**private Task Dequeue()**

**{**

**if (queue.Count > 0)**

**{**

**completedTaskCount += 1;**

**meanWorkTimeTask = workTime / completedTaskCount;**

**processingTimeSum += processingTime;**

**var createTime = queue.Peek().createTime;**

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

**meanWaitingTime = waitingTimeSum / completedTaskCount;**

**currentProcessingTime = null;**

**return queue.Dequeue();**

**}**

**else**

**{**

**return null;**

**}**

**}**

**public List<Task> Process(double progress)**

**{**

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

**if (IsEmpty()) {**

**downTime += progress;**

**return tasks;**

**}**

**workTime += progress;**

**time += progress;**

**// TODO: Fix calculation (сonsider progress)**

**queueCountSum += queue.Count;**

**processCount++;**

**meanQueueCount = queueCountSum / processCount;**

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

**{**

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

**// TODO: Handle time**

**if (currentProcessingTime < (time ))**

**{**

**time -= Convert.ToDouble(currentProcessingTime);**

**tasks.Add(Dequeue());**

**}**

**else**

**{**

**break;**

**}**

**}**

**if (IsEmpty()) time = 0;**

**return tasks;**

**}**

**public string GetTempStats()**

**{**

**string str = "";**

**str += "ЭВМ" + id + ": ";**

**str += "Очередь - " + queue.Count + ", ";**

**str += "Время обработки - " + Math.Round(currentProcessingTime ?? 0, 2) + " м., ";**

**str += "Процесс - " + Math.Round(time,2) + " м.\n";**

**return str;**

**}**

**public string GetStats()**

**{**

**string str = "";**

**str += "ЭВМ" + id + ":\n ";**

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

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

**str += "Коэффициент загрузки - " + Math.Round(workTime / (workTime + downTime > 0 ? workTime + downTime : 1), 2) + "\n ";**

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

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

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

**str += "Количество обработанных - " + completedTaskCount + "\n\n";**

**return str;**

**}**

**}**

**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 downTime = 0;**

**public double taskInterval = 0;**

**public int completedTaskCount = 0;**

**public int taskCount;**

**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)**

**{**

**if (completedTaskCount < settings.maxTasks)**

**{**

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

**{**

**downTime += progress;**

**}**

**else**

**{**

**workTime += progress;**

**}**

**}**

**var tasks1 = computer1.Process(progress);**

**for (int i = 0; i < tasks1.Count; i++)**

**{**

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

**if (taskType < settings.probMove2)**

**{**

**computer2.AddTask();**

**}**

**else**

**{**

**computer3.AddTask();**

**}**

**}**

**var tasks2 = computer2.Process(progress);**

**var tasks3 = computer3.Process(progress);**

**completedTaskCount += tasks2.Count + tasks3.Count;**

**if (taskCount < settings.maxTasks)**

**{**

**time += progress;**

**if(taskInterval <= time + 0.001)**

**{**

**while (taskInterval <= (time + 0.001) && taskCount < 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)**

**{**

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

**{**

**Process(taskInterval);**

**}**

**else**

**{**

**var min = Utils.min(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;**

**}**

**override**

**public string ToString()**

**{**

**string str = "";**

**str += "Временные параметры: \n";**

**str += computer1.GetTempStats();**

**str += computer2.GetTempStats();**

**str += computer3.GetTempStats();**

**str += "\nВыходные параметры: \n";**

**str += computer1.GetStats();**

**str += computer2.GetStats();**

**str += computer3.GetStats();**

**str += "Система:\n";**

**str += "Время работы: " + Math.Round(workTime, 2) + " м.\n";**

**str += "Время простоя: " + Math.Round(downTime, 2) + " м.\n";**

**double meanWorkTime = (computer1.workTime + computer2.workTime + computer3.workTime) / (computer1.completedTaskCount + computer2.completedTaskCount + computer3.completedTaskCount);**

**str += "Среднее время обработки задания - " + (meanWorkTime > 0 && !double.IsInfinity(meanWorkTime) ? Math.Round(meanWorkTime, 2) : 0) + "\n";**

**str += "Количество выполненных заданий: " + completedTaskCount + "\n";**

**return str;**

**}**

**}**