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 заданию на самоэкзамен**

**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](#_Toc131444650)

[**2 Краткое описание разработанного алгоритма** 3](#_Toc131444651)

**3** [**Приложение А Листинг программы** 9](#_Toc131444671)

**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**

Написанный код реализует алгоритм работы вычислительной системы, состоящей из трех компьютеров (Computer) и контролирующего класса ComputingSystem. Класс ComputingSystemSettings используется для хранения настроек системы.

Класс Task представляет задачу, которую нужно выполнить в вычислительной системе. Он имеет одно поле timeLeft, которое указывает, сколько времени занимает выполнение этой задачи. Если timeLeft равно null, значит для задачи ещё не подсчитано время.

Класс Computer представляет отдельный компьютер в системе, который имеет идентификатор id, очередь задач queue и время time, которое прошло с момента начала работы. Класс также содержит свойства, связанные с временем обработки processingTime и processingTimeError, и число выполненных задач completedTaskCount.

Метод IsEmpty() проверяет, пуста ли очередь задач на компьютере.

Метод AddTask() добавляет новую задачу в очередь.

Метод Enqueue(Task task) - добавляет задачу в конец очереди. Параметр task - задача, которую нужно добавить в очередь.

Dequeue() - извлекает и возвращает задачу из начала очереди. Если очередь пустая, то возвращает null. При извлечении задачи считает, что задача выполнена.

Peek() - возвращает задачу, которая находится в начале очереди, но не удаляет ее из очереди. Если очередь пустая, то возвращает null.

Метод Process(double progress) принимает на вход время progress, прошедшее с момента последнего вызова метода, и создает новые задачи с помощью метода AddTask() каждый раз, когда проходит заданный интервал времени taskInterval. Задачи добавляются в очередь на случайно выбранный компьютер в соответствии с вероятностями prob1 и prob2. Метод также обрабатывает задачи на каждом компьютере с помощью метода Process(double progress) и учитывает завершенные задачи.

Метод Restart() сбрасывает все свойства компьютера в начальное состояние.

Метод ToString() возвращает строковое представление состояния компьютера.

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 Restart() - сбрасывает состояние системы до начальных значений. Вызывает метод Restart() объектов Computer.

Метод InstantlyFinish() - мгновенно завершает все задачи в системе. Вызывает метод Process() с аргументом 1 до тех пор, пока очередь во всех компьютерах в системе не станет пуста.

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

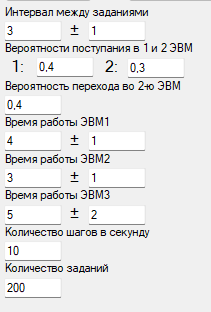
Таким образом, данная система реализует распределенную обработку задач между тремя компьютерами с заданными временными характеристиками. В конце работы программы выводится несколько параметров, а именно:

1. Количество выполненных заданий на всех трех компьютерах;
2. Общее количество выполненных заданий;
3. Общее время работы.

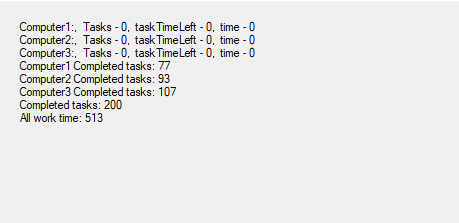
**4 Результаты экспериментов**

Эксперимент 1:

Изначальные параметры:

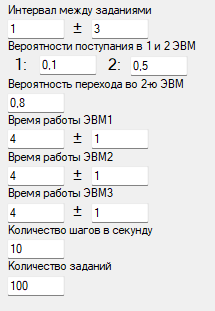
****

Результаты эксперимента:

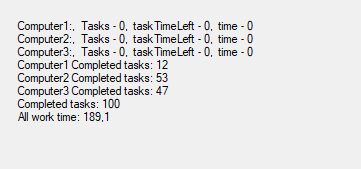


Эксперимент 2:

Начальные параметры:

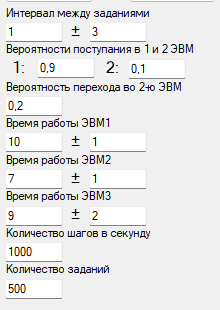
****

Результаты эксперимента:

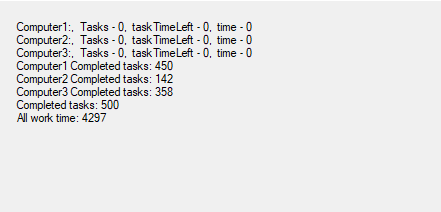


Эксперимент 3:

Изначальные параметры:

****

Результаты эксперимента:



**Приложение А  
Листинг программы**

**using System;**

**using System.Collections.Generic;**

**using System.Threading.Tasks;**

**public class Task**

**{**

**public int? timeLeft;**

**public Task(int? timeLeft)**

**{**

**this.timeLeft = timeLeft;**

**}**

**}**

**public class Computer**

**{**

**public int id;**

**public Queue<Task> queue;**

**public double time;**

**public int processingTime;**

**public int processingTimeError;**

**public int completedTaskCount;**

**private Random random;**

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

**{**

**this.id = id;**

**queue = new Queue<Task>();**

**time = 0;**

**this.processingTime = processingTime;**

**this.processingTimeError = processingTimeError;**

**random = new Random();**

**}**

**public bool IsEmpty()**

**{**

**return queue.Count == 0;**

**}**

**public void AddTask()**

**{**

**Task task = new Task(null);**

**Enqueue(task);**

**}**

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

**{**

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

**if (IsEmpty()) return tasks;**

**time += progress;**

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

**{**

**Task task = queue.Peek();**

**if (task.timeLeft == null) task.timeLeft = random.Next(processingTime - processingTimeError, processingTime + processingTimeError);**

**if (task.timeLeft < time)**

**{**

**time -= Convert.ToDouble(task.timeLeft) ;**

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

**} else**

**{**

**break;**

**}**

**}**

**time = Math.Round(time, 2);**

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

**return tasks;**

**}**

**private void Enqueue(Task task)**

**{**

**queue.Enqueue(task);**

**}**

**private Task Dequeue()**

**{**

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

**{**

**completedTaskCount += 1;**

**return queue.Dequeue();**

**}**

**else**

**{**

**return null;**

**}**

**}**

**private Task Peek()**

**{**

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

**{**

**return queue.Peek();**

**}**

**else**

**{**

**return null;**

**}**

**}**

**public void Restart()**

**{**

**queue.Clear();**

**time = 0;**

**completedTaskCount = 0;**

**}**

**override**

**public string ToString()**

**{**

**string str = "";**

**str += "Computer" + id + ":" + ", ";**

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

**str += "taskTimeLeft - " + (Peek()?.timeLeft ?? 0) + ", ";**

**str += "time - " + time + "\n";**

**return str;**

**}**

**}**

**public class ComputingSystemSettings**

**{**

**public int taskInterval;**

**public int taskIntervalError;**

**public double prob1;**

**public double prob2;**

**public int processingTime1;**

**public int processingTime1Error;**

**public int processingTime2;**

**public int processingTime2Error;**

**public int processingTime3;**

**public int processingTime3Error;**

**public int maxTasks;**

**public double probMove2;**

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

**{**

**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;**

**}**

**}**

**public class ComputingSystem**

**{**

**ComputingSystemSettings settings;**

**public Computer computer1;**

**public Computer computer2;**

**public Computer computer3;**

**private Random random;**

**public double allTime;**

**public double time;**

**public double timeLeft;**

**public int completedTaskCount;**

**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);**

**time = 0;**

**timeLeft = 0;**

**allTime = 0;**

**taskCount = 0;**

**completedTaskCount = 0;**

**random = new Random();**

**}**

**public void Process(double progress)**

**{**

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

**{**

**time += progress;**

**while (timeLeft < time && taskCount < settings.maxTasks) {**

**time -= timeLeft;**

**double taskType = random.NextDouble();**

**if (taskType < settings.prob1)**

**{**

**computer1.AddTask();**

**taskCount++;**

**}**

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

**{**

**computer2.AddTask();**

**taskCount++;**

**}**

**else**

**{**

**computer3.AddTask();**

**taskCount++;**

**}**

**timeLeft = random.Next(settings.taskInterval - settings.taskIntervalError, settings.taskInterval + settings.taskIntervalError);**

**}**

**}**

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

**{**

**allTime += progress;**

**allTime = Math.Round(allTime,2);**

**}**

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

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

**{**

**double taskType = random.NextDouble();**

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

**{**

**computer2.AddTask();**

**}**

**else**

**{**

**computer3.AddTask();**

**}**

**}**

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

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

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

**}**

**public void Restart()**

**{**

**computer1.Restart();**

**computer2.Restart();**

**computer3.Restart();**

**time = 0;**

**taskCount = 0;**

**completedTaskCount = 0;**

**}**

**public void InstantlyFinish()**

**{**

**while (true)**

**{**

**Process(1);**

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

**{**

**break;**

**}**

**}**

**}**

**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 += computer1.ToString();**

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

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

**str += "Сomputer1 Completed tasks: " + computer1.completedTaskCount + "\n";**

**str += "Сomputer2 Completed tasks: " + computer2.completedTaskCount + "\n";**

**str += "Сomputer3 Completed tasks: " + computer3.completedTaskCount + "\n";**

**str += "Completed tasks: " + completedTaskCount + "\n";**

**str += "All work time: " + allTime + "\n";**

**return str;**

**}**

**}**