

федеральное государственное бюджетное образовательное учреждение   
высшего образования   
«Санкт-Петербургский государственный технологический институт  
(технический университет)»  
СПбГТИ(ТУ)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| УГНС | | 09.00.00 | Информатика и вычислительная техника | | | |
| Направление подготовки | | 09.03.03 | Прикладная информатика в химии | | | |
| Направленность (профиль) | |  | САПР | | | |
|  | |  |  | | | |
| Факультет | |  | Информационных технологий и управления | | | |
| Кафедра | |  | Систем автоматизированного  проектирования и управления | | | |
| Учебная дисциплина | |  | Методы поддержки принятия решений | | | |
|  |  | | |  |  |  |

Курс \_\_\_3\_\_\_ Группа 475

**Лабораторная работа №4**

**Теория массового обслуживания**

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Андрианова К.И.\_\_

(подпись, дата) (инициалы, фамилия)

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Овчинников Р.С.

(подпись, дата) (инициалы, фамилия)

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Пекер В.А.

(подпись, дата) (инициалы, фамилия)

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Уланов В.Н.

(подпись, дата) (инициалы, фамилия)

Санкт-Петербург

2019

# Цель работы

Целью данной работы является создание программы, которая осуществляет решение по теории массового обслуживания. Предлагается практическое решение задачи. Для удобства использования программы разработан эргономичный интерфейс.

**План выполнения**

1. Знакомство с теорией массового обслуживания
2. Блок-схема программы
3. Разработка интерфейса
4. Тестирование
5. Анализ тестирования
6. Вывод

**Ход выполнения работы**

1. **Теория массового обслуживания**, или **очередей** — раздел [теории вероятностей](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%B5%D0%B9), целью исследований которого является рациональный выбор структуры системы обслуживания и процесса обслуживания на основе изучения потоков требований на обслуживание, поступающих в систему и выходящих из неё, длительности ожидания и длины [очередей](https://ru.wikipedia.org/wiki/%D0%9E%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D1%8C). В теории массового обслуживания используются методы теории вероятностей и [математической статистики](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0).

Теория массового обслуживания заключается в том, что на базе теории вероятностей выводятся математические методы анализа процессов массового обслуживания, а также методы оценки качества работы обслуживающих систем.

При всем своём разнообразии процессы в системах массового обслуживания имеют общие черты:

Требование на обслуживание не регулярно случайно поступает на **канал обслуживания** и в зависимости от его занятости, продолжительности обслуживания образуют очередь требований.

Теория массового обслуживания изучает статистические закономерности поступления. И на этой основе вырабатывает решения, то есть такие характеристики системы обслуживания, при которых затраты времени на ожидание в очереди и на каналы обслуживания были бы наименьшими. (если мало каналов обслуживания — то образуются большие очереди, и наоборот, если много каналов обслуживания, то очередей нет, но при этом каналы обслуживания работают не рационально, так как часть из них простаивает без работы).

Теория массового обслуживания — это прикладная область теории случайных процессов.

**Предметом** исследования **теории массового обслуживания** являются **вероятностные модели физических систем обслуживания**, в которых случайные и не случайные моменты времени возникают заявки на обслуживание и имеются устройства на обработку данных заявок.

Теория массового обслуживания целиком базируется на теории вероятности и на математической статистике. В определенной степени она связана с распределением Пуассона, которое описывает вероятность числа появлений в заданном интервале времени какого-либо события. Например, появление покупателя у прилавка, если известно, что появление события зависит от того давно ли оно появлялось в последний раз и сколько раз и когда именно случалось до этого.

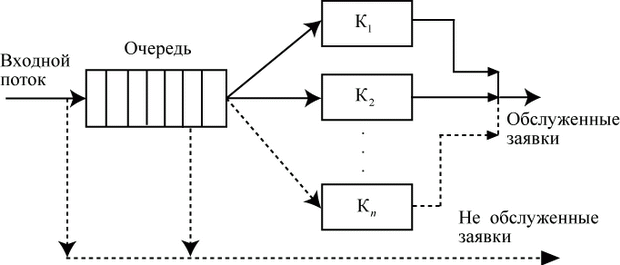
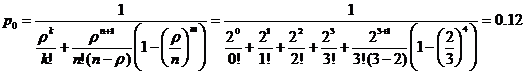


Рисунок 1 – Алгоритм работы теории массового обслуживания

**Среднее время ожидания обслуживания заявки в очереди**

[Исчисляем показатели обслуживания многоканальной СМО](https://math.semestr.ru/cmo/mcmo.php) (онлайн):  
Интенсивность потока обслуживания:  
https://www.semestr.ru/images/math/cmo/c10_image001.gif  
1. Интенсивность нагрузки.  
ρ = λ • tобс = 120 • 1/60 = 2  
 Интенсивность нагрузки ρ=2 показывает степень согласованности входного и выходного потоков заявок канала обслуживания и определяет устойчивость системы массового обслуживания.  
3. Вероятность, что канал свободен (доля времени простоя каналов).  
  
Следовательно, 12% в течение часа канал будет не занят, время простоя равно tпр = 7.1 мин.  
Вероятность того, что обслуживанием:  
занят 1 канал:  
p1  = ρ1/1! p0 = 21/1! • 0.12 = 0.24  
заняты 2 канала:  
p2  = ρ2/2! p0 = 22/2! • 0.12 = 0.24  
заняты 3 канала:  
p3  = ρ3/3! p0 = 23/3! • 0.12 = 0.16  
4. Доля заявок, получивших отказ.  
https://www.semestr.ru/images/math/cmo/c10_image003.gif  
Значит, 3% из числа поступивших заявок не принимаются к обслуживанию.

5. Вероятность обслуживания поступающих заявок.  
В системах с отказами события отказа и обслуживания составляют полную группу событий, поэтому:  
pотк + pобс = 1  
Относительная пропускная способность: Q = pобс.  
pобс = 1 - pотк = 1 - 0.0311 = 0.97  
Следовательно, 97% из числа поступивших заявок будут обслужены. Приемлемый уровень обслуживания должен быть выше 90%.  
6. Среднее число каналов, занятых обслуживанием.  
nз = ρ • pобс = 2 • 0.97 = 1.9 каналов  
Среднее число простаивающих каналов.  
nпр = n - nз = 3 - 1.9 = 1.1 каналов  
7. Коэффициент занятости каналов обслуживанием.  
https://www.semestr.ru/images/math/cmo/c10_image004.gif  
Следовательно, система на 60% занята обслуживанием.  
8. Абсолютная пропускная способность.  
A = pобс • λ = 0.97 • 120 = 116.3 заявок/час.  
9. Среднее время простоя СМО.  
tпр = pотк • tобс = 0.0311 • 0.0166 = 0 час.  
10. Среднее число заявок, находящихся в очереди.  
https://www.semestr.ru/images/math/cmo/c10_image005.gif  
https://www.semestr.ru/images/math/cmo/c10_image006.gif ед.  
11. Среднее время простоя СМО (среднее время ожидания обслуживания заявки в очереди).  
Среднее время простоя СМО: формула час.  
12. Среднее число обслуживаемых заявок.  
Lобс = ρ • Q = 2 • 0.97 = 1.94 ед.  
13. Среднее число заявок в системе.  
LCMO = Lоч + Lобс = 0.51 + 1.94 = 2.45 ед.  
13. Среднее время пребывания заявки в СМО.  
https://www.semestr.ru/images/math/cmo/c10_image008.gif час.  
 Число заявок, получивших отказ в течение часа: λ • p1 = 4 заявок в  час.  
Номинальная производительность СМО: 3 / 0.0166 = 181 заявок в час.  
Фактическая производительность СМО: 116.3 / 181 = 64% от номинальной производительности.

**Выбор инструментария**

Программа написана в Visual Studio 2017 на языке C#.

**Интерфейс**

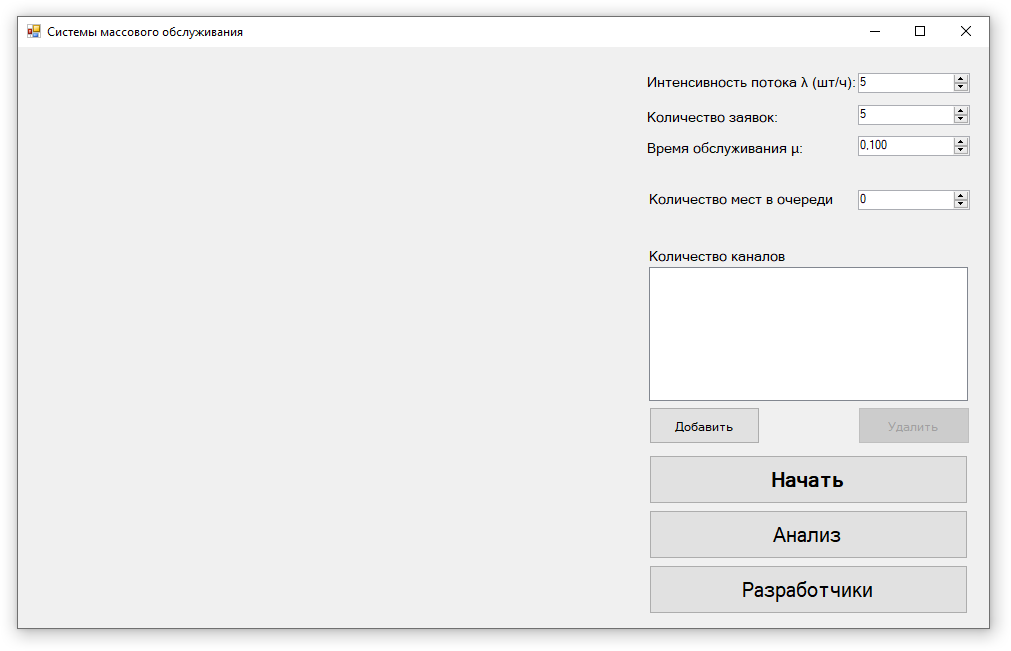
****

Рисунок 2 – Интерфейс программы

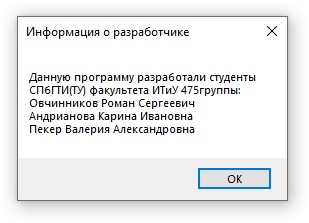
****

Рисунок 3 – Информация о разработчиках

**Блок-схемы программы**

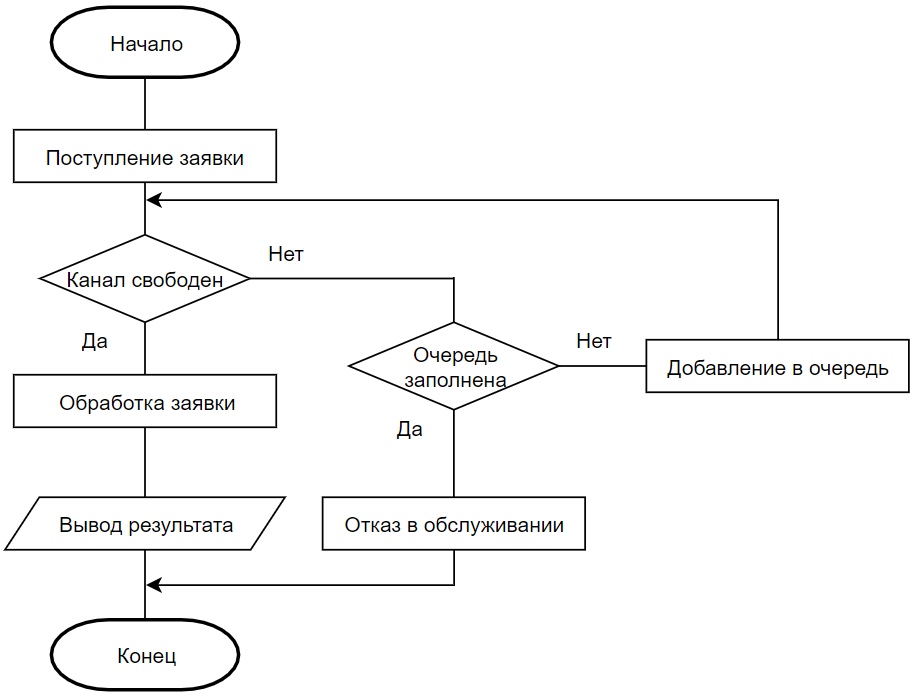


Рисунок 4 – Блок-схема программы

**Тестирование**

Сначала мы заполняем поля интенсивность потока, время обслуживания, максимальная очередь, ограничение по заявкам и количество каналов. (рис.5).

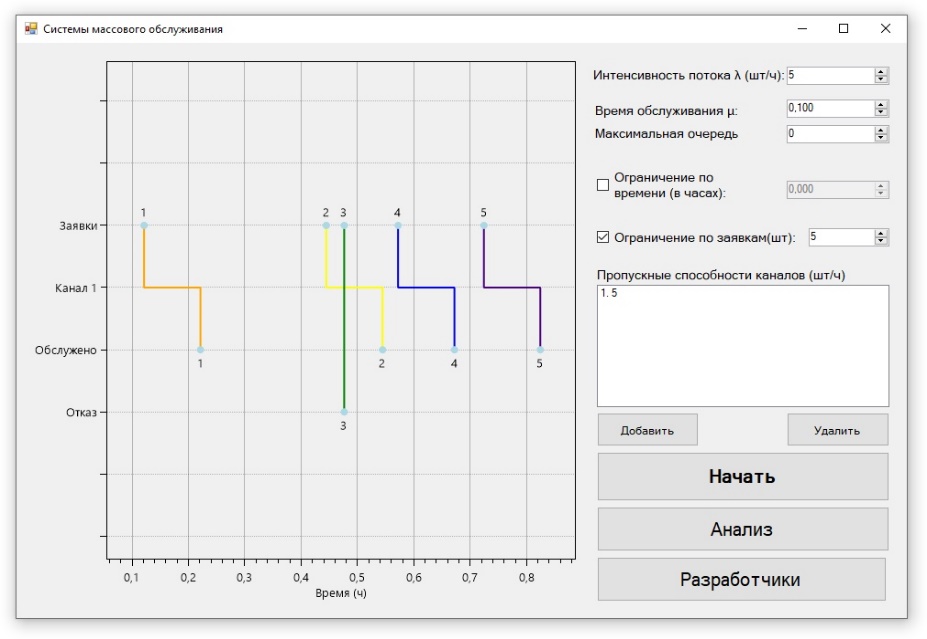


Рисунок 5 – Заполнение всех полей

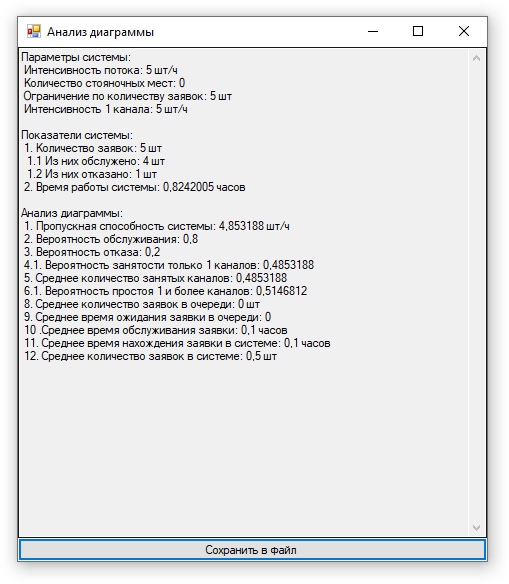


Рисунок 6 – Вывод анализа программы

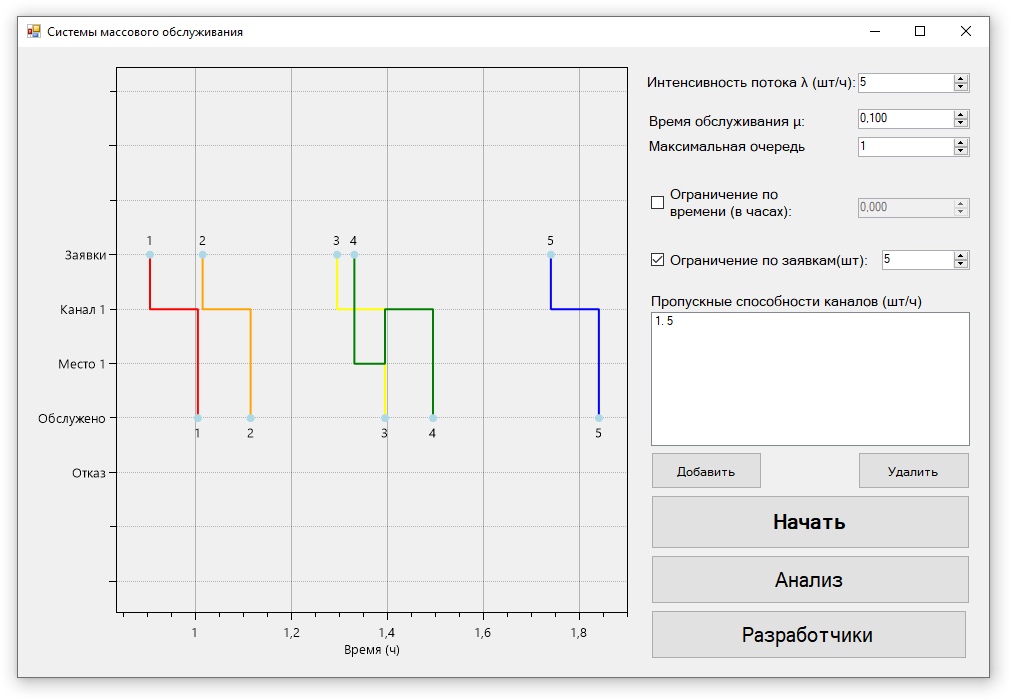
Далее мы проводим проверку работы программы в случае, если добавляем места в очереди. На (рис.8,9) видим, что система работает безотказно

Рисунок 7 – Результат программы при добавлении мест в очередь

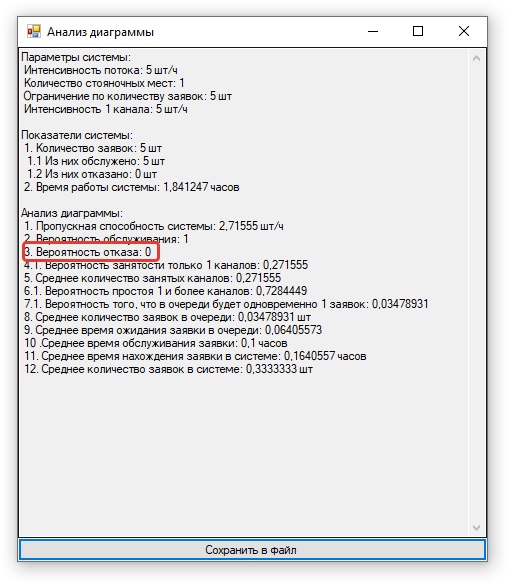


Рисунок 8 – Результат программы при добавлении мест в очередь

Далее, мы проводим тестирование, где увеличиваем количество заявок, и наблюдаем, что система становится неустойчива (рис. 10, 11)

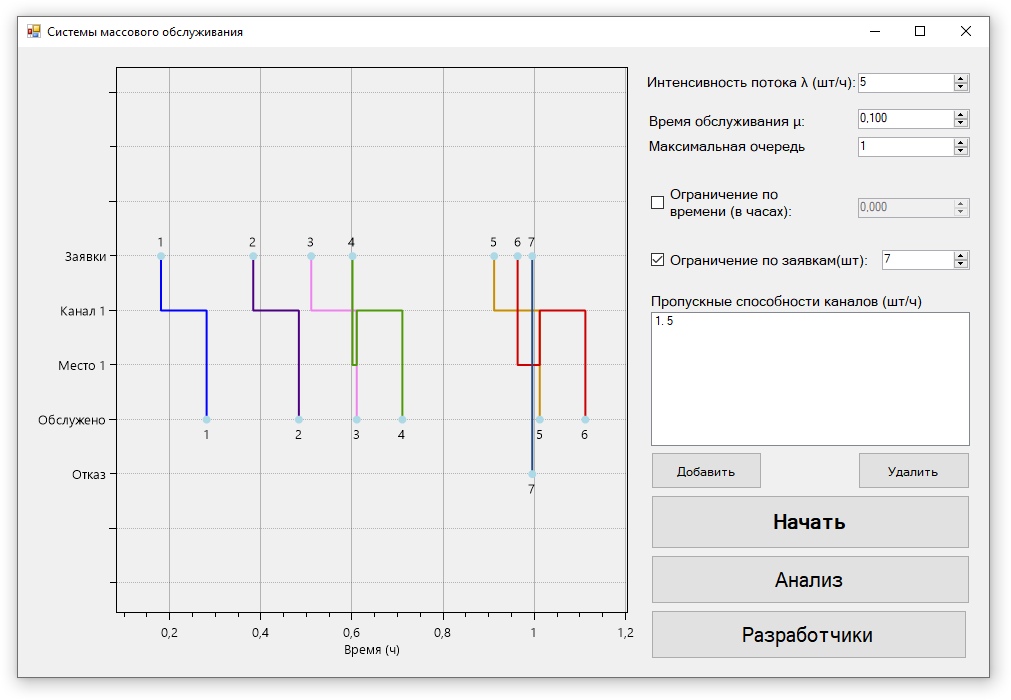


Рисунок 9 – Результат программы при увеличении количества заявок

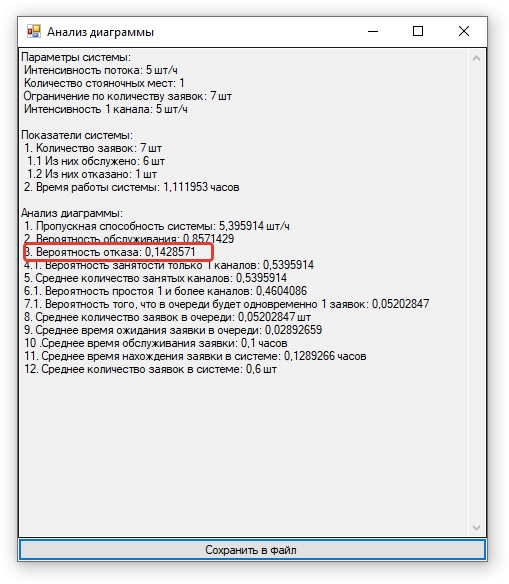
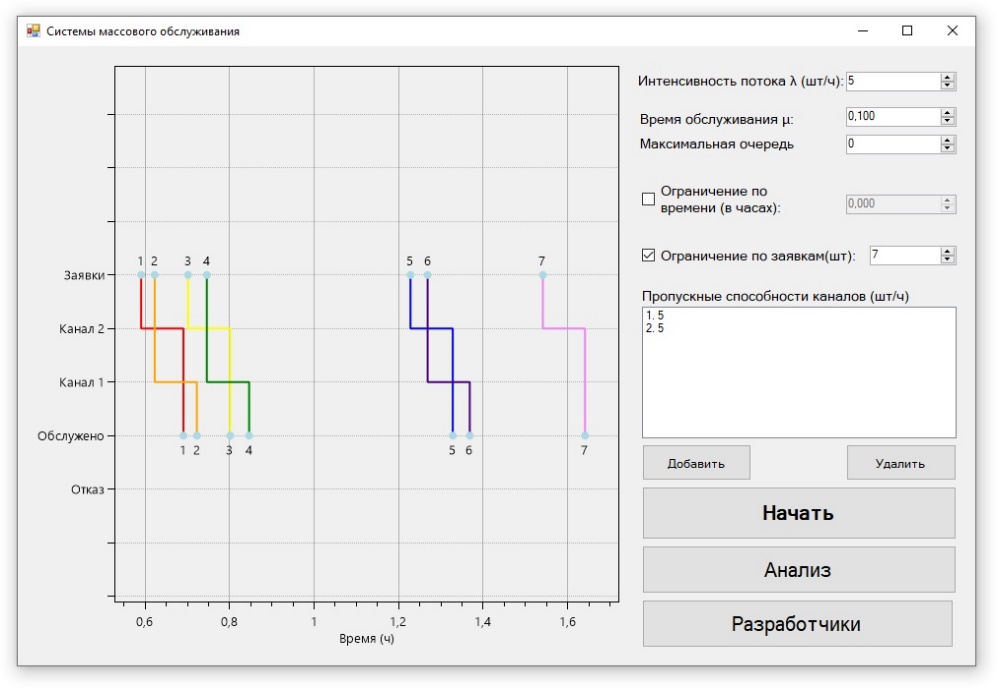


Рисунок 10 – Результат программы при увеличении количества заявок

Чтобы решить данную проблемы, мы увеличили количество запросов. Система вновь становится устойчивой. (рис 12,13)



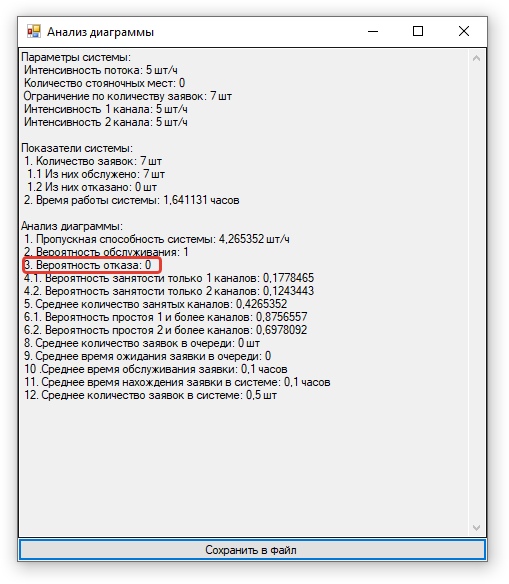
Рисунок 11 – Результат программы при увеличении количества заявок и запросов

Рисунок 12 – Результат программы при увеличении количества заявок и запросов

В случае, если не будут введены каналы, всё уйдёт в отказ. (рис.14)

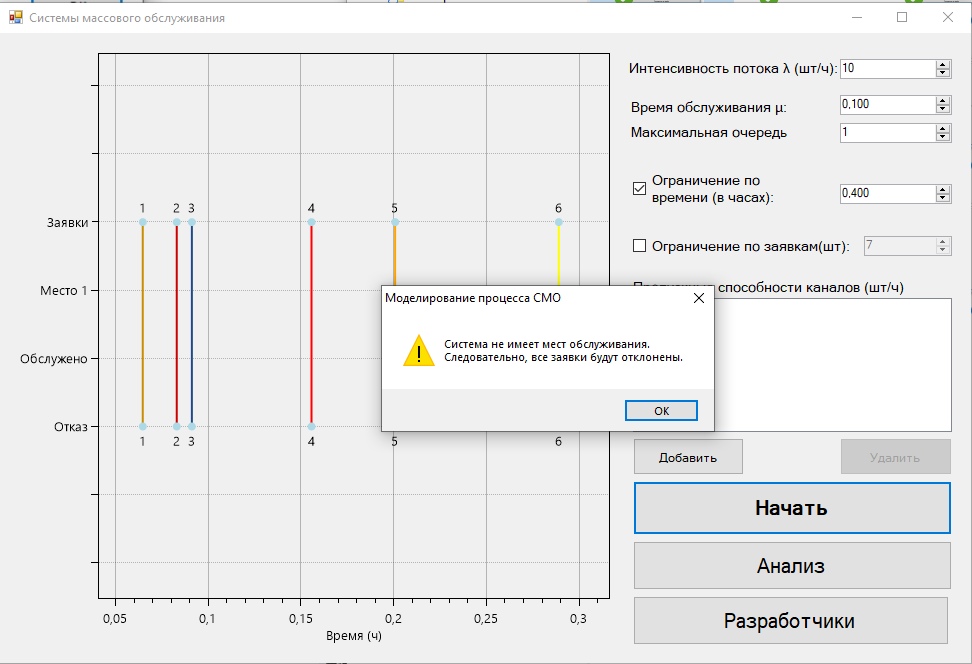


Рисунок 13 – Результат программы если не ввести каналы.

**Анализ тестирования**

В ходе тестирования мы убедились, что написанная нами программа работает. Интерфейс, предложенный нами эргономичен и удобен для использования.

**Код программы (алгоритм)**

/// Моделирует процесс СМО.

class ProcessModeller

{

ParametersContainer Parameters { get; }

double[] channelIdleTimes { get; }

double[] queueIdleTimes { get; }

public ProcessModeller(ParametersContainer parameters)

{

Parameters = parameters;

channelIdleTimes = new double[Parameters.ChannelCount];

queueIdleTimes = new double[parameters.QueueCapacity];

for (int i = 0; i < Parameters.ChannelCount; i++)

channelIdleTimes[i] = -1;

for (int i = 0; i < Parameters.QueueCapacity; i++)

queueIdleTimes[i] = -1;

}

/// Возвращает индекс доступного свободного места в указанный момент времени. Если такого нет, возвращает -1.

int GetQueuePlaceIndex(double arrivalTime)

{

for (int i = 0; i < Parameters.QueueCapacity; i++)

{

if (queueIdleTimes[i] < arrivalTime)

return i;

}

return -1;

}

/// Возвращает индекс доступного места обслуживания в указанный момент времени. Если такого нет, возвращает -1.

int GetNextPossibleChannel(double arrivalTime)

{

if (Parameters.ChannelCount == 0)

return -1;

int minIndex = 0;

double minimumValue = channelIdleTimes[0];

for (int i = 0; i < Parameters.ChannelCount; i++)

{

if (Parameters.PreferFirstChannel && channelIdleTimes[i] < arrivalTime)

return i;

if (channelIdleTimes[i] < minimumValue)

{

minimumValue = channelIdleTimes[i];

minIndex = i;

}

}

return minIndex;

}

/// Создает новую линию в пути заявки, обслуженной на указанном месте обслуживания.

void CreateChannelLine(int channelIndex, double arrivalTime, RandomGenerator rnd, TimeDiagram timeDiagram)

{

double realRndValue= 1;

double clientSerivceTime = (double)Parameters.NumericUpDown1;

double departureTime = arrivalTime + clientSerivceTime;

channelIdleTimes[channelIndex] = departureTime;

timeDiagram.PushChannelLine(channelIndex, clientSerivceTime, realRndValue);

timeDiagram.PushServedPoint();

}

/// Добавляет в систему новую заявку и строит ее путь.

void PushClient(double arrivalTime, RandomGenerator rnd, double realRndValue, double interval, TimeDiagram timeDiagram)

{

timeDiagram.PushStartPoint(arrivalTime, realRndValue, interval);

int usingChannel = GetNextPossibleChannel(arrivalTime);

if (usingChannel == -1)

{

timeDiagram.PushRefusedPoint();

return;

}

if (channelIdleTimes[usingChannel] < arrivalTime)

{

CreateChannelLine(usingChannel, arrivalTime, rnd, timeDiagram);

return;

}

int queuePlaceIndex = GetQueuePlaceIndex(arrivalTime);

if (queuePlaceIndex == -1)

{

timeDiagram.PushRefusedPoint();

return;

}

double queuedStart = arrivalTime;

for (int i = queuePlaceIndex; i >= 0; i--)

{

double queuedEndTime = i != 0 ? queueIdleTimes[i - 1] : channelIdleTimes[usingChannel];

queueIdleTimes[i] = queuedEndTime;

timeDiagram.PushQueueLine(i, queuedEndTime - queuedStart);

queuedStart = queuedEndTime;

}

CreateChannelLine(usingChannel, queueIdleTimes[0], rnd, timeDiagram);

}

void FillDiagram(TimeDiagram timeDiagram)

{

double step = 0.5 / Parameters.ThreadIntencity;

RandomGenerator rnd = new RandomGenerator(Parameters.MinRandomValue);

double realTime = 0;

double fixedTime = 0;

for (int i = 0; !Parameters.HasClientLimit || i < Parameters.ClientLimit; i++)

{

double realRndValue;

double rndValue = rnd.Next(Parameters.ThreadIntencity, out realRndValue);

realTime += rndValue;

while (fixedTime < realTime)

{

timeDiagram.CheckClientCountAtTime(fixedTime);

fixedTime += step;

}

if (Parameters.HasTimeLimit && realTime > Parameters.TimeLimit)

break;

PushClient(realTime, rnd, realRndValue, rndValue, timeDiagram);

}

timeDiagram.FinishDiagram();

}

public TimeDiagram CreateDiagram()

{

var diagram = new TimeDiagram(Parameters.ChannelCount, Parameters.QueueCapacity);

FillDiagram(diagram);

return diagram;

}

public TimeDiagram CreateDiagram<T>(IGraphicsFactory<T, IGraph> factory, out InteractiveDiagram interactiveDiagram) where T : InteractiveDiagram

{

List<string> labels = new List<string>(Parameters.ChannelCount + Parameters.QueueCapacity + 3);

labels.Add("Заявки");

for (int i = 0; i < Parameters.ChannelCount; i++)

labels.Add($"Канал { Parameters.ChannelCount - i }");

for (int i = 0; i < Parameters.QueueCapacity; i++)

labels.Add($"Место { Parameters.QueueCapacity - i }");

labels.Add("Обслужено");

labels.Add("Отказ");

T idg = factory.CreateEmptyDiagram(labels);

TimeDiagram diagram = new TimeDiagram(Parameters.ChannelCount, Parameters.QueueCapacity, idg);

FillDiagram(diagram);

interactiveDiagram = idg;

return diagram;

}

}

**Вывод**

В ходе лабораторной работы мы изучили теорию массового обслуживания. Написали программу, с помощью которой мы выясняем при заданных условиях, какая вероятность обслуживания, пропускная способность системы, вероятность отказа, вероятность занятости каналов, среднее количество заявок в очереди, среднее время ожидания заявки, среднее время обслуживания заявки.