Министерство науки высшего образования

Российской Федерации

Новосибирский государственный технический университет

Кафедра Автоматизированных систем управления

Лабораторная работа №1

**Построение и оценка качества генераторов псевдослучайных чисел**

Факультет: АВТФ Преподаватель:

Группа: АВТ-113 Лыгина Н.И.

Студенты:

Зеленев П.А.

Осокин Д.М.

Салихова Е.А.

Новосибирск 2023

Оглавление

[**1.** **Цель** 3](#_Toc131951310)

[**2.** **Вариант задания. Конвейерная система.** 3](#_Toc131951311)

[**3.** **Показатели эффективности функционирования системы.** 3](#_Toc131951312)

[**4.** **Представление системы в виде Q-схемы.** 3](#_Toc131951313)

[**5.** **Концептуальная модель (моделирующий алгоритм).** 3](#_Toc131951314)

[**6.** **Иерархическая структура программных модулей.** 3](#_Toc131951315)

[**7.** **Программная реализация модели.** 3](#_Toc131951316)

[**8.** **Результаты моделирования.** 3](#_Toc131951317)

[**9.** **Выводы.** 3](#_Toc131951318)

1. **Цель**

Изучение имитационного подхода в моделировании на примере типовых

математических схем систем массового обслуживания (Q – схем).

1. **Вариант задания. Конвейерная система.**

Конвейерная линия состоит из 5 обслуживающих устройств, расположенных вдоль конвейера. Детали поступают на обработку в соответствии с равномерным законом распределения с интенсивностью 4 единицы в минуту. Продолжительность обслуживания на каждом устройстве распределена экспоненциально с математическим ожиданием 1 мин. Свободного места перед каждым устройством нет, поэтому устройство может снять деталь с конвейера, только если находится в состоянии "СВОБОДЕН". Деталь на обработку поступает к свободному устройству, и по окончании покидает систему. Интервал исследования детали между устройствами равен 1 мин. Оцените, насколько улучшится работа конвейера, если перед каждым устройством будет буфер для одной детали.

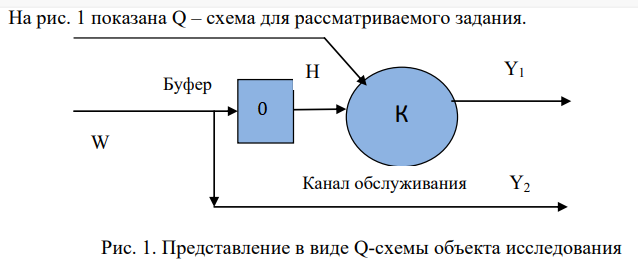
1. **Показатели эффективности функционирования системы.**

вероятность того, что требование будет обслужено ;

среднее время обслуживания заявки ;

среднее время простоя каналов ;

1. **Представление системы в виде Q-схемы.**

****

1. **Концептуальная модель (моделирующий алгоритм).**
2. **Иерархическая структура программных модулей.**
3. **Программная реализация модели.**

|  |
| --- |
| класс HandlerOfRequests.cs |
| using System;  using System.Collections.Generic;  using System.Data.SqlTypes;  using System.IO;  namespace conveyorSystem  {  public class HandlerOfRequests  {  public double RequestProssesingTime { get; set; }  private int modelingTime;  List<double> reqestInQueue = new List<double>();  private int queueSize;  private bool end;  private Random rnd;  public HandlerOfRequests(int modelingTime, int queueSize, Random rnd)  {  this.rnd = rnd;  end = false;  this.modelingTime = modelingTime;  if (queueSize < 0)  throw new Exception("размер очереди не может быть меньше нуля");  this.queueSize = queueSize;  RequestProssesingTime = 0;  totalDownTime = 0;  totalWaitingTime = 0;  totalProcessingTime = 0;  counterOfServedRequests = 0;  }  public bool IsFreeWhenProsessing(double ArrivalRequestTime)  {  // удаление из очереди заявки если пришло время ее обработки  if (reqestInQueue.Count == queueSize && queueSize != 0)  {  if (ArrivalRequestTime >= reqestInQueue[0])  {  reqestInQueue.RemoveAt(0);  }  }  // если обработчик свободен, обрабатываем заявку  if (ArrivalRequestTime > RequestProssesingTime && ArrivalRequestTime < modelingTime)  {  if (end == true)  return false;  // обработать случай конца моделирования  totalDownTime += ArrivalRequestTime - RequestProssesingTime;  StartProsessing(ArrivalRequestTime);  return true;  }  // если обработчик занят, добавляем заявку в очередь  else if (reqestInQueue.Count != queueSize && queueSize != 0)  {  if (end == true)  return false;  totalWaitingTime += RequestProssesingTime - ArrivalRequestTime;  reqestInQueue.Add(RequestProssesingTime);  StartProsessing(reqestInQueue[0]);  return true;  }  else  return false;  }  private void StartProsessing(double ArrivalRequestTime)  {    // время когда освободиться тот или иной обработчик  double randomValue = rnd.NextDouble();  double nnn = -(1.0 \* 60) \* Math.Log(randomValue);  RequestProssesingTime = nnn + ArrivalRequestTime;    if (RequestProssesingTime > modelingTime)  end = true;  else  {  totalProcessingTime += nnn;  counterOfServedRequests++;  }  }  public void FinalCollectionOfStatistics()  {  if (counterOfServedRequests > 0)  {  AverageDownTime = Math.Round(totalDownTime / counterOfServedRequests / 60, 2);  AverageWaitingTime = Math.Round(totalWaitingTime / counterOfServedRequests / 60, 2);  AverageProcessingTime = Math.Round(totalProcessingTime / counterOfServedRequests / 60, 2);  }  }  private double totalDownTime;  private double totalWaitingTime;  private double totalProcessingTime;  private int counterOfServedRequests;  public double AverageDownTime { get; private set; }  public double AverageWaitingTime { get; private set; }  public double AverageProcessingTime { get; private set; }  }  } |

|  |
| --- |
| класс Model.cs |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data.SqlTypes;  using System.Diagnostics;  namespace conveyorSystem  {  public class Model  {  public List<HandlerOfRequests> Handlers { get; private set; }  public int modelingTime;  private double arrivalTimeOfRequest;  private double requestProsessingTime;  private const int OneMinute = 60;  private int counterOfRefusal;  private int numberOfHandlers;  private int queueSize;  private Random rnd;  public Model(int modelingTime, int queueSize, int numberOfHandlers, Random rnd)  {  this.rnd = rnd;  if (modelingTime <= 0)  throw new Exception("время не может быть отрицательным");  this.modelingTime = modelingTime \* OneMinute;  if (queueSize < 0)  throw new Exception("размер очереди не может быть отрицательным");  this.queueSize = queueSize;  if (numberOfHandlers <= 0)  throw new Exception("количество обработчиков не может быть отрицательным");  this.numberOfHandlers = numberOfHandlers;  arrivalTimeOfRequest = 0;  requestProsessingTime = 0;  counterOfRefusal = 0;  Handlers = new List<HandlerOfRequests>();  }  public void Modeling()  {  // Создание обработчиков заявок  for (int i = 0; i < numberOfHandlers; i++)  Handlers.Add(new HandlerOfRequests(modelingTime, queueSize, rnd));  int counterOfRequests = 0;  while (true)  {  // создание времени прихода заявки в секундах  double randomValue = rnd.NextDouble();  const int intensivnostPerMinute = 4;  arrivalTimeOfRequest += randomValue \* OneMinute / intensivnostPerMinute;    // подсчёт пришедших заявок  counterOfRequests++;  // заканчиваем моделирование  if (arrivalTimeOfRequest > modelingTime)  break;  // обработка заявки  bool flagOfProcessing = false;  for (int i = 0; i < Handlers.Count; i++)  {  double arrivalTime = arrivalTimeOfRequest;  if (Handlers[i].IsFreeWhenProsessing(arrivalTimeOfRequest))  {  flagOfProcessing = true;  break;  }  arrivalTime += OneMinute;  }    // считаем количество отказов  if (!flagOfProcessing)  {  counterOfRefusal++;  }  }  FalureProbability = Math.Round((double)counterOfRefusal/counterOfRequests, 2);  ServiceProbability = 1 - FalureProbability;  for (int i = 0; i < Handlers.Count; i++)  {  Handlers[i].FinalCollectionOfStatistics();  }  // ModelingDone.Invoke();  }  // public event Action ModelingDone;  public double ServiceProbability { get; private set; }  public double FalureProbability { get; private set; }  }  } |

|  |
| --- |
| класс Form1.cs |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace conveyorSystem  {  public partial class Form1 : Form  {  private Model model;  public Form1()  {  InitializeComponent();  ServiceProbability = new List<double>();  FalureProbability = new List<double>();  AverageProcessingTime = new List<List<double>>();  AverageDownTime = new List<List<double>>();  AverageWaitingTime = new List<List<double>>();  }  private void button1\_Click(object sender, EventArgs e)  {  try  {  Random rnd = new Random();  int N = 50, \_N = 51;  double kvantil = 1.96;  double tochnost = Convert.ToDouble(textBox1.Text);  while (\_N > N)  {  N = \_N;  for (int i = 0; i < N; i++)  {  model = new Model(Convert.ToInt32(modelingTime.Text), Convert.ToInt32(queueSize.Text), Convert.ToInt32(countOfHeandlers.Text),rnd);  // model.ModelingDone += UploadForm;  model.Modeling();  AverageProcessingTime.Add(new List<double>());  AverageDownTime.Add(new List<double>());  AverageWaitingTime.Add(new List<double>());  FindStatistics(i);  }  double servProb = 0;  for (int i = 0; i < ServiceProbability.Count; i++)  {  servProb += ServiceProbability[i];  }  \_N = (int)(((servProb / N)\*(1 - servProb/N) \* Math.Pow(kvantil,2))/Math.Pow(tochnost,2));  if (\_N > N)  {  ClearForm();  }  }  UploadForm(N);  ClearForm();  }  catch (Exception ex)  {  MessageBox.Show(ex.Message);  modelingTime.Clear();  }  }  private void ClearForm()  {  ServiceProbability.Clear();  FalureProbability.Clear();  AverageProcessingTime.Clear();  AverageDownTime.Clear();  AverageWaitingTime.Clear();  }  private void UploadForm(int n)  {  double servProb = 0, falureProb = 0;  List<double> AverageProcessingT = new List<double>();  List<double> AverageDownT = new List<double>();  List<double> AverageWaitingT = new List<double>();  for (int i = 0; i < AverageDownTime[0].Count; i++)  {  AverageProcessingT.Add(0);  AverageDownT.Add(0);  AverageWaitingT.Add(0);  }  dataGridView1.Rows.Clear();  for (int i = 0; i < ServiceProbability.Count; i++)  {  servProb += ServiceProbability[i];  falureProb += FalureProbability[i];  for (int j = 0; j < AverageDownTime[i].Count; j++)  {  AverageProcessingT[j] += AverageProcessingTime[i][j];  AverageDownT[j] += AverageDownTime[i][j];  AverageWaitingT[j] += AverageWaitingTime[i][j];  }  }  serviceProbability.Text = $"Вероятность обслуживания {Math.Round(servProb/n,2)}";  falureProbability.Text = $"Вероятность отказа {1 - Math.Round(servProb/n,2)}";    var heandlers = model.Handlers;  for (int i = 0; i < heandlers.Count; i++)  {  dataGridView1.Rows.Add(i + 1, Math.Round(AverageProcessingT[i]/n, 2),  Math.Round(AverageDownT[i] / n, 2),  Math.Round(AverageWaitingT[i] / n,2));  }  }  private void FindStatistics(int n)  {  ServiceProbability.Add(model.ServiceProbability);  FalureProbability.Add(model.FalureProbability);  var heandlers = model.Handlers;  for (int i = 0; i < heandlers.Count; i++)  {  AverageProcessingTime[n].Add(heandlers[i].AverageProcessingTime);  AverageDownTime[n].Add(heandlers[i].AverageDownTime);  AverageWaitingTime[n].Add(heandlers[i].AverageWaitingTime);  }  }  private List<double> ServiceProbability;  private List<double> FalureProbability;  private List<List<double>> AverageProcessingTime;  private List<List<double>> AverageDownTime;  private List<List<double>> AverageWaitingTime;  private void Label5\_Click(object sender, EventArgs e)  {  }  }  } |

1. **Результаты моделирования.**

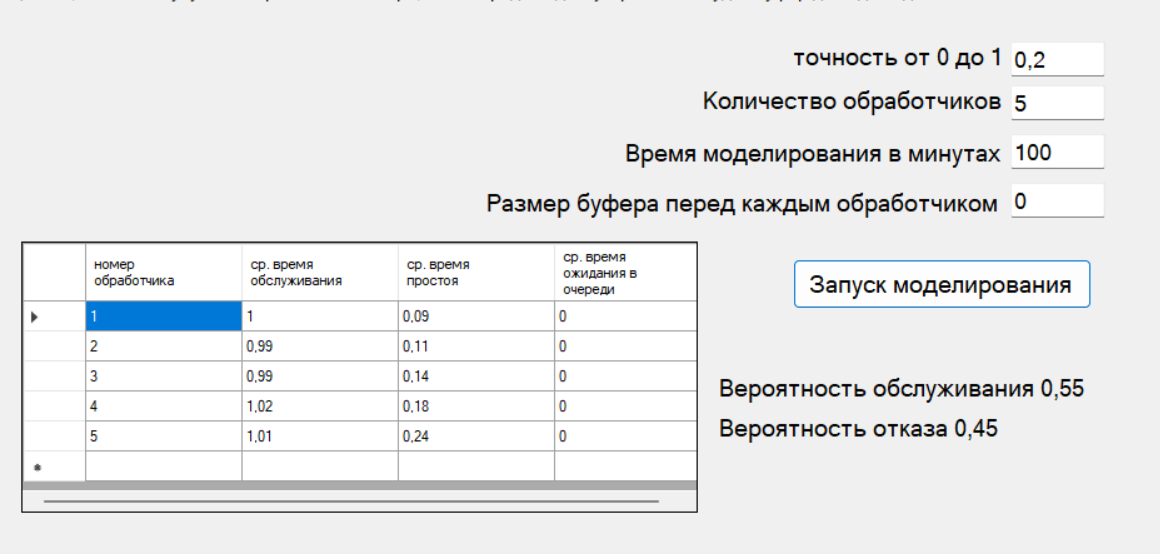
****

Рис.2 – моделирование СМО без буферов перед каждым обработчиком

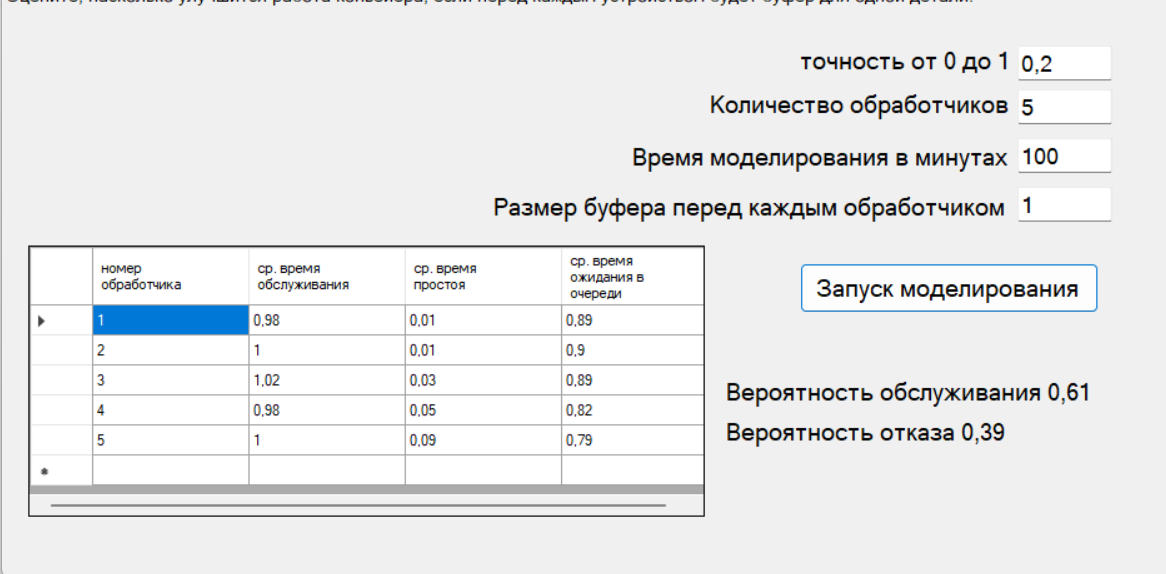
****

Рис.3 – моделирование СМО с буфером размером 1 перед каждым обработчиком

1. **Выводы.**

Благодаря увеличению размера буфера перед каждым устройством, растет вероятность обслуживания заявки на 6%, среднее время простоя каждого из устройств снижается до 0,01 мин; 0,01 мин; 0,03 мин; 0,05 мин; 0,09 мин; соответственно и появляется показатель среднее время ожидания в очереди, так как появляется буфер.