Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №5 по дисциплине "Моделирование"

Тема Моделирование работы информационного центра

Студент Малышев И. А.

Группа ИУ7-71Б

Оценка (баллы) _____

Преподаватель: Рудаков И. В.

Москва — 2022 г.

1 | Задание

В информационный центр приходят клиенты через интервал времени 10 ± 2 минуты. Если все три имеющихся оператора заняты, клиенту отказывают в обслуживании. Операторы имеют разную производительность и могут обеспечивать обслуживание среднего запроса пользователя за 20 ± 5; 40 ± 10; 40 ± 20. Клиенты стремятся занять свободного оператора с максимальной производительностью. Полученные запросы сдаются в накопитель. Откуда выбираются на обработку. На первый компьютер запросы от 1 и 2-ого операторов, на второй – запросы от 3-его. Время обработки запросов первым и 2- м компьютером равны соответственно 15 и 30 мин. Промоделировать процесс обработки 300 запросов.

Для выполнения поставленного задания необходимо создать концептуальную модель в терминах СМО, определить эндогенные и экзогенные переменные и уравнения модели. За единицу системного времени выбрать 0.01 минуты.

2 | Решение

2.1 Теоретическая часть

2.1.1 Концептуальная модель системы в терминах СМО

На рисунке 2.1 предоставлена концептуальная модель моделируемой системы в терминах СМО.

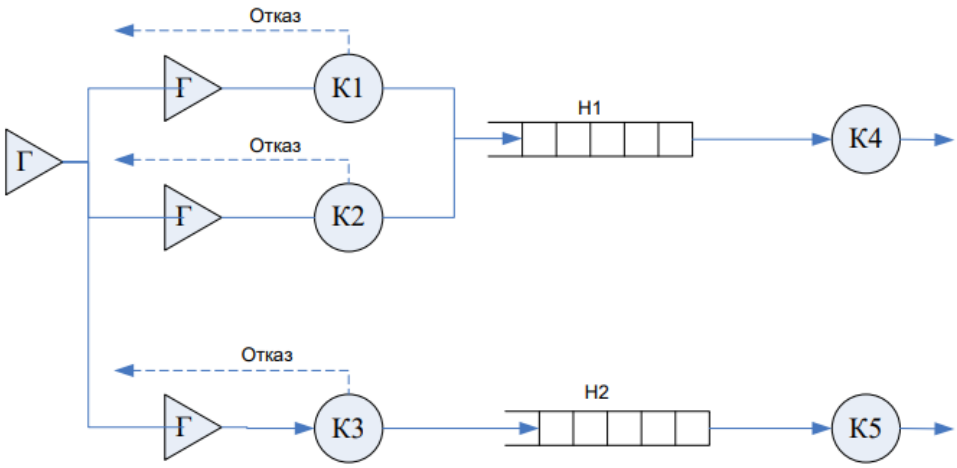

Рис. 2.1: Концептуальная модель системы в терминах СМО.

В процессе взаимодействия клиентов с информационным центром возможен.

1. Режим нормального обслуживания, т.е. клиент выбирает одного из свободных операторов, отдавая предпочтение тому у которого меньше номер.

2. Режим отказа в обслуживании клиента, когда все операторы заняты.

2.1.2 Переменные и уравнения имитационной модели

Эндогенные переменные: время обработки задания i-ым оператором, время решения этого задания j-ым компьютером.

Экзогенные переменные: число обслуженных клиентов и число клиентов, получивших отказ.

Вероятность отказа в обслуживании клиента:

$$P = \frac{C_{refus}}{C_{refus} + C_{serv}}, \tag{2.1}$$

где C_{refus} – количество заявок, которым было отказано в обслуживании, C_{serv} – – количество заявок, которые были обслужены.

2.2 Листинг

Далее представлен фрагмент программы, выполняющий поставленное задание.

```
using Generator = Func<double>;

internal interface Processor
{
  bool GetRequest();
}

static class ModelTimer
{
  public static double CurrentTime { get; set; } = 0;
}

class RequestGenerator
{
  Generator generator;
  public int numberOfGeneratedRequests;
  public double timeOfNextEvent;
  List<Processor> receivers;

  public RequestGenerator(Generator generator)
  {
    this.generator = generator;
    numberOfGeneratedRequests = 0;
    receivers = new List<Processor>();
    timeOfNextEvent = 0;
  }

  public void SubscribeReceiver(Processor receiver)
  {
    receivers.Add(receiver);
  }

  public double GetNextTime() => generator();

  public Processor SendRequest()
```

```
36    {
37      numberOfGeneratedRequests++;
38      foreach (var receiver in receivers)
39      {
40        if (receiver.GetRequest())
41        return receiver;
42      }
43
44      return null;
45    }
46  }
47
48  class RequestProcessor: RequestGenerator, Processor
49  {
50    public int numberOfRequestsInQueue, numberOfProcessedRequests;
51    int maxOfQueue;
52
53    public RequestProcessor(Generator generator, int maxOfQueue = 0) :
         base(generator)
54    {
55      numberOfRequestsInQueue = 0;
56      numberOfProcessedRequests = 0;
57      this.maxOfQueue = maxOfQueue;
58    }
59
60    public void Process()
61    {
62      if (numberOfRequestsInQueue > 0)
63      {
64        numberOfProcessedRequests++;
65        numberOfRequestsInQueue--;
66        SendRequest();
67      }
68
69      timeOfNextEvent = numberOfRequestsInQueue > 0 ? ModelTimer.CurrentTime
           + GetNextTime() : 0.0;
70    }
71
72    public bool GetRequest()
73    {
74      if (maxOfQueue == 0 || numberOfRequestsInQueue < maxOfQueue)
75      {
76        numberOfRequestsInQueue++;
77        if (timeOfNextEvent == 0)
78        timeOfNextEvent = ModelTimer.CurrentTime + GetNextTime();
79
80        return true;
81      }
82      else
83      return false;
```

```
84    }
85  }
86
87  struct SimulationParameters
88  {
89    public Client client;
90    public Operator operator1, operator2, operator3;
91    public Computer computer1, computer2;
92
93    public SimulationParameters(Client client, Operator operator1, Operator
        operator2, Operator operator3, Computer computer1, Computer computer2)
94    {
95      this.client = client;
96      this.operator1 = operator1;
97      this.operator2 = operator2;
98      this.operator3 = operator3;
99      this.computer1 = computer1;
100     this.computer2 = computer2;
101   }
102 }
103
104 struct Client
105 {
106   public int timeOfCome, timeDelta, amountOfProccessedNeeded;
107
108   public Client(int timeOfCome, int timeDelta, int
        amountOfProccessedNeeded)
109   {
110     this.timeDelta = timeDelta;
111     this.timeOfCome = timeOfCome;
112     this.amountOfProccessedNeeded = amountOfProccessedNeeded;
113   }
114 }
115
116 struct Operator
117 {
118   public int timeOfService, timeDelta;
119
120   public Operator(int timeOfService, int timeDelta)
121   {
122     this.timeOfService = timeOfService;
123     this.timeDelta = timeDelta;
124   }
125 }
126
127 struct Computer
128 {
129   public int timeOfService;
130
131   public Computer(int timeOfService)
```

```
132   {
133     this.timeOfService = timeOfService;
134   }
135 }
136 
137 struct Results
138 {
139   public int numberOfDenials;
140   public double probabilityOfDenial;
141 
142   public Results(int numberOfDenials, double probabilityOfDenial)
143   {
144     this.numberOfDenials = numberOfDenials;
145     this.probabilityOfDenial = probabilityOfDenial;
146   }
147 
148   public static Results DoSimulate(SimulationParameters parameters)
149   {
150     RequestGenerator generatorOfClients = new RequestGenerator(() =>
151     UniformDistributions.GetUniformReal(
152     parameters.client.timeOfCome - parameters.client.timeDelta,
153     parameters.client.timeOfCome + parameters.client.timeDelta));
154 
155     RequestProcessor operator1 = new RequestProcessor(() =>
156     UniformDistributions.GetUniformReal(
157     parameters.operator1.timeOfService - parameters.operator1.timeDelta,
158     parameters.operator1.timeOfService + parameters.operator1.timeDelta),
          1);
159 
160     RequestProcessor operator2 = new RequestProcessor(() =>
161     UniformDistributions.GetUniformReal(
162     parameters.operator2.timeOfService - parameters.operator2.timeDelta,
163     parameters.operator2.timeOfService + parameters.operator2.timeDelta),
          1);
164 
165     RequestProcessor operator3 = new RequestProcessor(() =>
166     UniformDistributions.GetUniformReal(
167     parameters.operator3.timeOfService - parameters.operator3.timeDelta,
168     parameters.operator3.timeOfService + parameters.operator3.timeDelta),
          1);
169 
170     RequestProcessor computer1 = new RequestProcessor(() =>
          parameters.computer1.timeOfService);
171     RequestProcessor computer2 = new RequestProcessor(() =>
          parameters.computer2.timeOfService);
172 
173     generatorOfClients.SubscribeReceiver(operator1);
174     generatorOfClients.SubscribeReceiver(operator2);
175     generatorOfClients.SubscribeReceiver(operator3);
176 
```

```
177     operator1.SubscribeReceiver(computer1);
178     operator2.SubscribeReceiver(computer1);
179     operator3.SubscribeReceiver(computer2);
180
181     RequestGenerator[] schemeElements = new RequestGenerator[]{
182       generatorOfClients, operator1, operator2, operator3, computer1,
            computer2};
183
184     int numberOfDenials = 0;
185     generatorOfClients.timeOfNextEvent = generatorOfClients.GetNextTime();
186
187     while (computer1.numberOfProcessedRequests +
           computer2.numberOfProcessedRequests <
188     parameters.client.amountOfProccessedNeeded)
189     {
190       ModelTimer.CurrentTime = generatorOfClients.timeOfNextEvent;
191       foreach (var element in schemeElements)
192       {
193         if (element.timeOfNextEvent != 0 && element.timeOfNextEvent <
              ModelTimer.CurrentTime)
194         {
195           ModelTimer.CurrentTime = element.timeOfNextEvent;
196         }
197       }
198
199       foreach (var element in schemeElements)
200       {
201         if (ModelTimer.CurrentTime == element.timeOfNextEvent)
202         {
203           RequestProcessor processor = element as RequestProcessor;
204           if (processor is not null)
205             processor.Process();
206           else
207           {
208             Processor catchProcessor = generatorOfClients.SendRequest();
209             if (catchProcessor is null)
210             numberOfDenials++;
211
212             generatorOfClients.timeOfNextEvent = ModelTimer.CurrentTime +
                    generatorOfClients.GetNextTime();
213           }
214         }
215       }
216     }
217
218     return new Results(numberOfDenials, numberOfDenials /
          ((double)numberOfDenials + operator1.numberOfProcessedRequests +
219     operator2.numberOfProcessedRequests +
220     operator3.numberOfProcessedRequests));
221   }
```

```
222 }
223
224 static class UniformDistributions
225 {
226   public static double GetUniformReal(double aParameter, double bParameter)
227   {
228     return ContinuousUniform.Sample(aParameter, bParameter);
229   }
230
231   public static int GetUniformInt(int aParameter, int bParameter)
232   {
233     return (int)Math.Round(ContinuousUniform.Sample(aParameter,
          bParameter));
234   }
235 }
```

2.3 Результаты работы

Моделирование проводилось с использованием событийного принципа.

Важно отметить, что в силу того, что система подразбита на два «домена», вычисляемая вероятность отказа прямо не зависит от скорости обработки запросов компьютеров, которая позволяют лишь увеличить или уменьшить время моделирования системы, так как конечным условием моделирования является обработка 300 запросов. В качестве количества обслуженных заявок бралась сумма обработанных операторами заявок.

На рисунке 2.2 представлен пользовательский интерфейс программы в исходном состоянии.

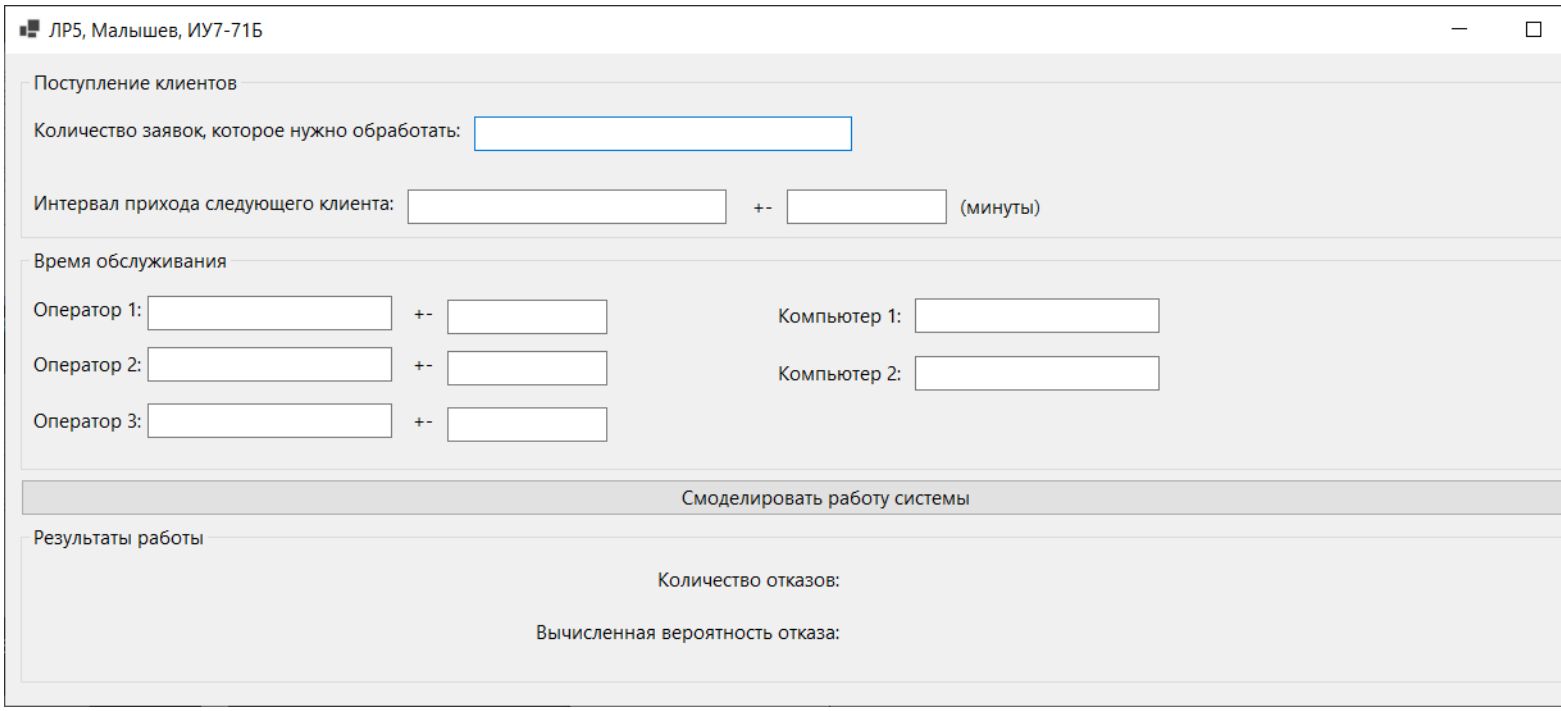

Рис. 2.2: Пользовательский интерфейс программы в исходном состоянии.

На рисунках 2.3-2.4 представлены примеры результатов работы программы с указанными данными.

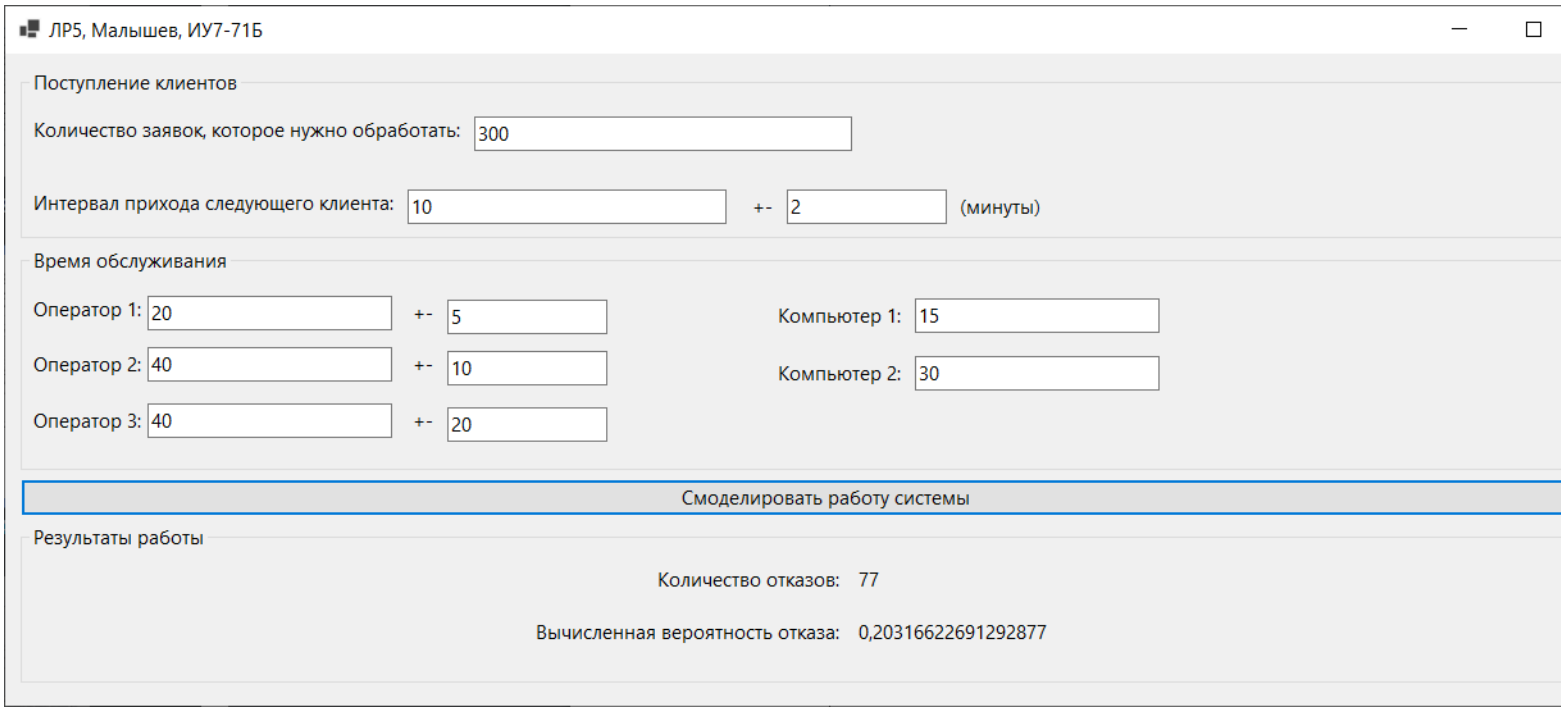

Рис. 2.3: Пример работы реализованного приложения.

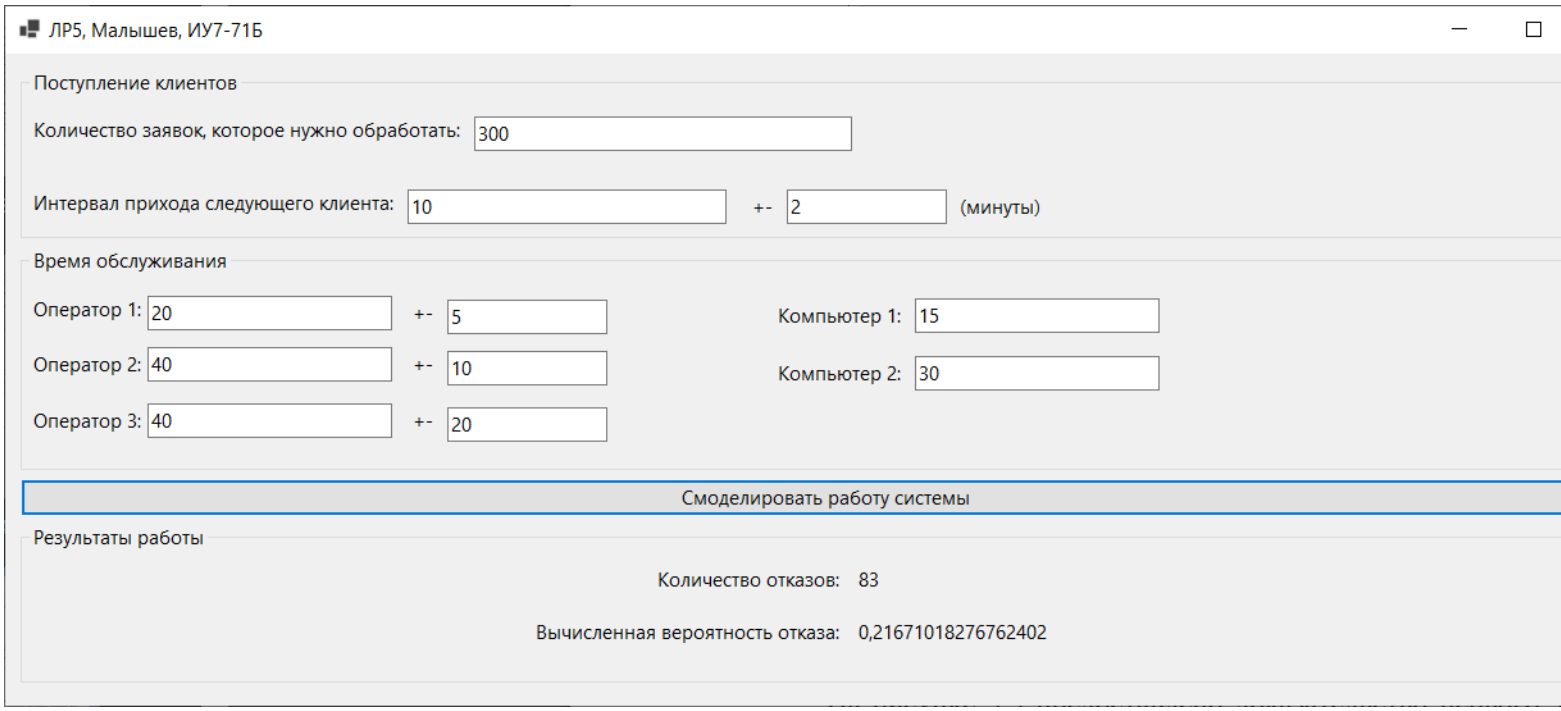

Рис. 2.4: Пример работы реализованного приложения.

На рисунке 2.5 предоставлено доказательство первого замечания. Как видно, вычисленная вероятность отказа остаётся примерно такой же, как и на предыдущих данных.

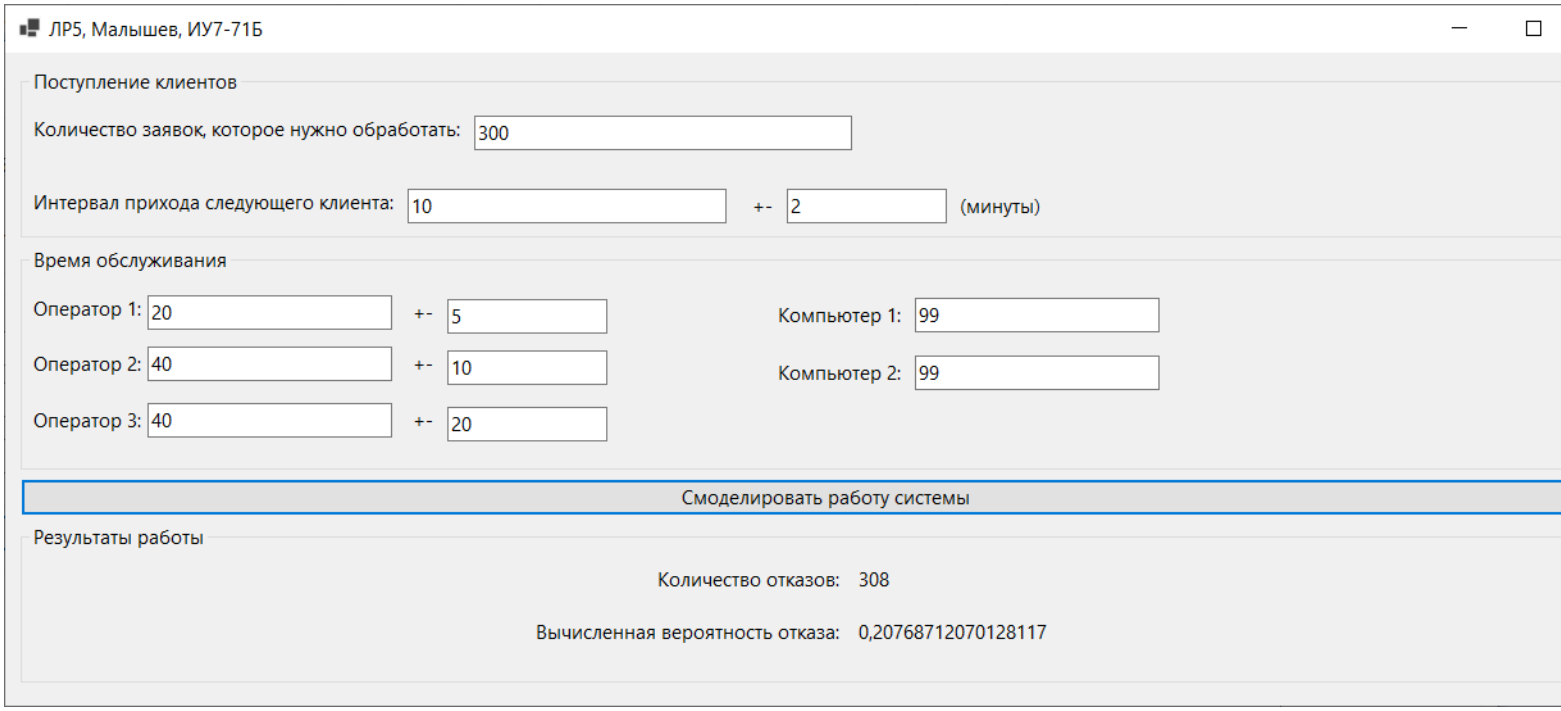

Рис. 2.5: Значительное уменьшение скорости обработки запросов компьютерами.