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»

Отчёт

по лабораторной работе №4

Название: Обслуживающий аппарат

Дисциплина: Моделирование

Студент ИУ7-75Б _____ Д.В. Сусликов

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподователь _____ И.В. Рудаков

(Подпись, дата) (И.О. Фамилия)

Москва, 2021

Задание

Необходимо промоделировать систему, состоящую из генератора, памяти, и обслуживающего аппарата. Генератор подает сообщения, распределенные по равномерному закону, они приходят в память и выбираются на обработку по закону из ЛР1 (распределение Пуассона). Количество заявок конечно и задано. Предусмотреть случай, когда обработанная заявка возвращается обратно в очередь. Необходимо определить оптимальную длину очереди, при которой не будет потерянных сообщений. Реализовать двумя способами: используя пошаговый и событийный подходы.

Теория

Равномерное распределение

Функция распределения:

$$F_X(x) \equiv P(X \le x) = \begin{cases} 0, & x < a \\ \dfrac{x-a}{b-a}, & a \le x < b \\ 1, & x \ge b \end{cases}.$$

Функция плотности распределения:

$$f_X(x) = \begin{cases} \frac{1}{b-a}, & x \in [a,b] \\ 0, & x \notin [a,b] \end{cases}.$$

Пуассоновское распределение

Функция распределения:

$$F(x) = \frac{\Gamma(|k+1), \lambda)}{|k|!}$$

Плотность распределения:

$$f(x) = \frac{\lambda^k}{k!} e^{-\lambda}$$

Пошаговый подход

Заключается в последовательном анализе состояний всех блоков системы в момент $t + \Delta t$. Новое состояние определяется в соответствии с их алгоритмическим описанием с учетом действия случайных факторов. В результате этого анализа принимается решение о том, какие системные события должны имитироваться на данный момент времени. Основной недостаток: значительные затраты и опасность пропуска события при больших Δt.

Событийная модель

Состояния отдельных устройств изменяются в дискретные моменты времени. При использовании событийного принципа, состояния всех блоков системы анализируются лишь в момент возникновения какого либо события. Момент наступления следующего события, определяется минимальным значением из списка событий.

Результаты работы

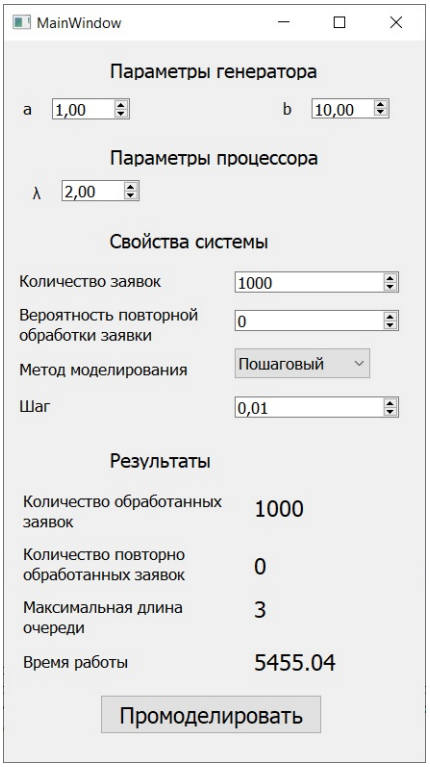

Рисунок 1 – Пошаговый при p = 0

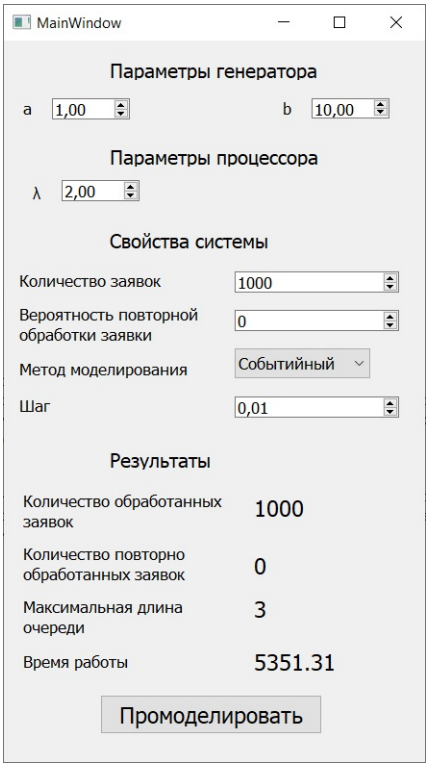

Рисунок 2 – Событийный при p = 0

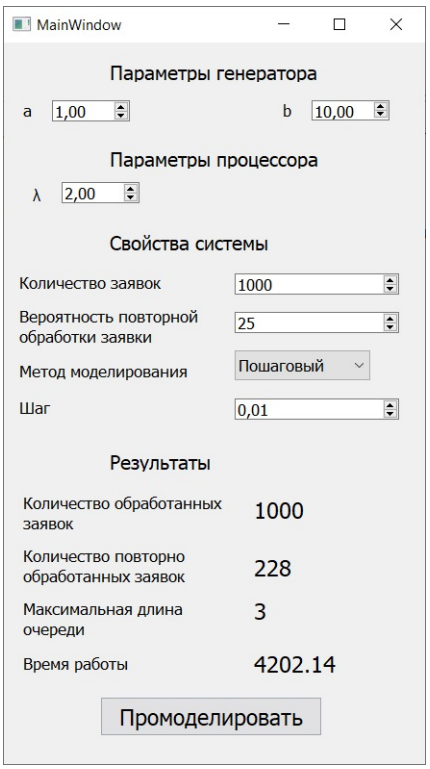

Рисунок 3 – Пошаговый при p = 25

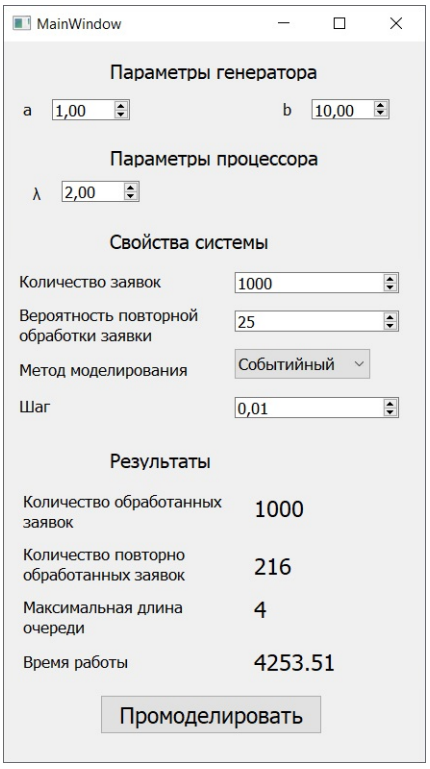

Рисунок 4 – Событийный при p = 25

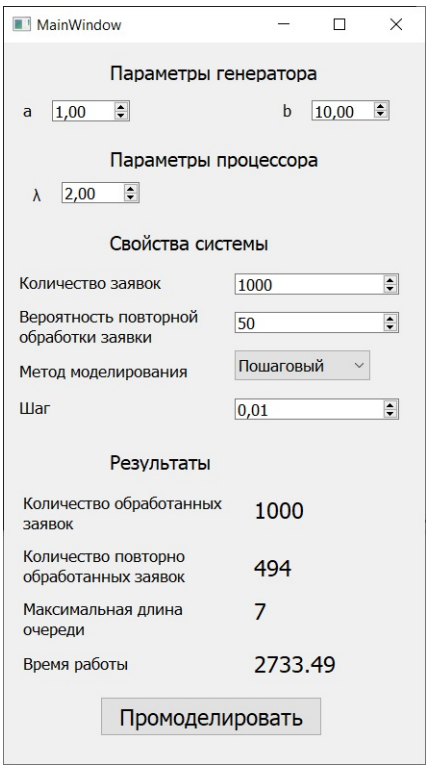

Рисунок 5 – Пошаговый при p = 50

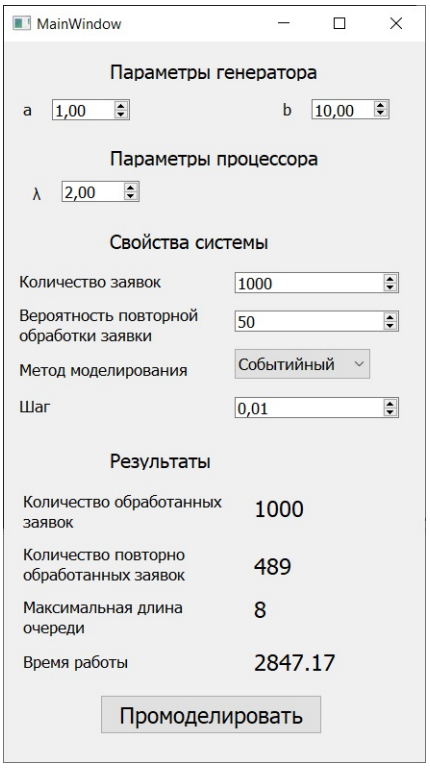

Рисунок 6 – Событийный при p = 50

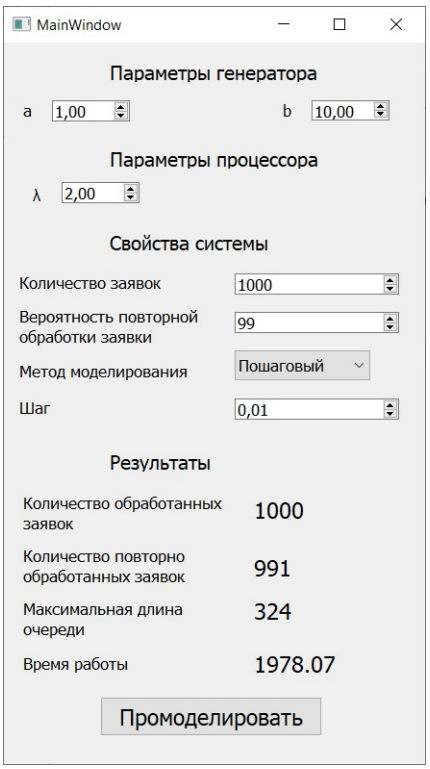

Рисунок 7 – Пошаговый при p = 99

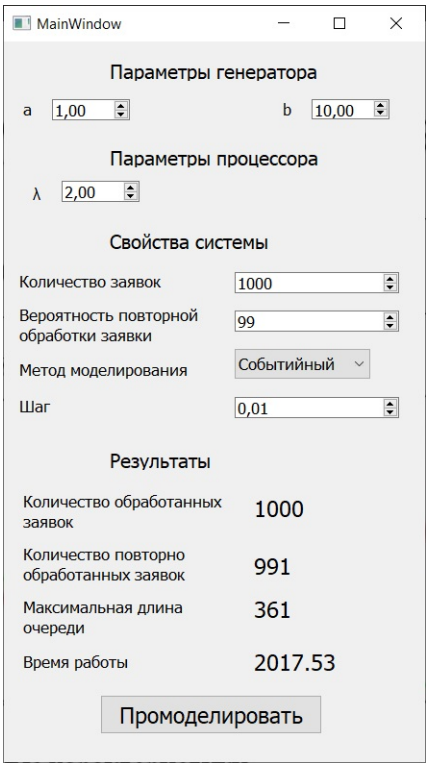

Рисунок 8 – Событийный при p = 99

Вывод

Была смоделирована система, состоящая из генератора, памяти и обслуживающего аппарата.

На выходе получена максимальная длина очереди, число обрабатываемых и повторно обработанных заявок, время обработки.

Листинг

Листинг 1 – main.py

```
import sys
from PyQt5 import QtWidgets
from PyQt5 import uic, QtWidgets, QtGui
from PyQt5.QtWidgets import QApplication, QWidget, QListWidgetItem
    , QTableWidgetItem, QMessageBox
import design
from models import event_model, step_model
from distributions import UniformDistribution, PoissonDistribution

class App(QtWidgets.QMainWindow, design.Ui_MainWindow):
    def __init__(self):
        a, b = 1, 10
        self.generator = UniformDistribution(a, b)

        lyambda = 2.0
        self.processor = PoissonDistribution(lyambda)

        self.total_tasks = 1000
        self.repeat_percentage = 0
        self.step = 0.01

        super().__init__()
        self.setupUi(self)
        self.initUI()
```

```
24
25      def initUI(self):
26          self.startBtn.clicked.connect(self.startBtnPushed)
27
28
29      def startBtnPushed(self):
30          a = self.aSpinBox.value()
31          b = self.bSpinBox.value()
32          lyambda = self.lyambdaSpinBox.value()
33
34          self.generator = UniformDistribution(a, b)
35          self.processor = PoissonDistribution(lyambda)
36
37          self.total_tasks = self.tasksSpinBox.value()
38          self.repeat_percentage = self.repeatSpinBox.value()
39          self.step = self.stepSpinBox.value()
40          processed_tasks, reentered_tasks, max_queue_len, t = 0, 0,
                0, 0
41          if self.methodComboBox.currentText() == "Sobitiyniy":
42              processed_tasks, reentered_tasks, max_queue_len, t =
                    event_model(self.generator, self.processor, self.
                    total_tasks, self.repeat_percentage)
43          else:
44              processed_tasks, reentered_tasks, max_queue_len, t =
                    step_model(self.generator, self.processor, self.
                    total_tasks, self.repeat_percentage)
45
46          self.resTasksLabel.setText(str(processed_tasks))
47          self.repeatTasksLabel.setText(str(reentered_tasks))
48          self.maxLenLabel.setText(str(max_queue_len))
49          self.timeLabel.setText(str(round(t, 2)))
50
51
52  def main():
53      app = QtWidgets.QApplication(sys.argv)
```

```
54      window = App()
55      window.show()
56      app.exec_()
57
58      print('event_model:', event_model(generator, processor,
            total_tasks, repeat_percentage))
59      print('step_model:', step_model(generator, processor,
            total_tasks, repeat_percentage, step))
60
61
62  if __name__ == '__main__':
63      main()
```

Листинг 2 – models.py

```
1   import random
2
3
4   def step_model(generator, processor, total_tasks=0, repeat=0, step
        =0.001):
5       processed_tasks = 0
6       t_curr = step
7       t_gen_prev = t_proc = 0
8       t_gen = generator.generate()
9       t_proc = t_gen + processor.generate()
10      cur_queue_len = max_queue_len = 0
11      reentered_tasks = 0
12
13      while processed_tasks < total_tasks:
14      if t_curr > t_gen:
15          cur_queue_len += 1
16          if cur_queue_len > max_queue_len:
17              max_queue_len = cur_queue_len
18          t_gen += generator.generate()
19
20          if t_curr > t_proc:
```

```
            if cur_queue_len > 0:
                processed_tasks += 1
            if random.randint(1, 100) <= repeat:
                cur_queue_len += 1
                reentered_tasks += 1
            cur_queue_len -= 1
            if cur_queue_len > 0:
                t_proc += processor.generate()
            else:
                t_proc = t_gen + processor.generate()
    t_curr += step

    return processed_tasks, reentered_tasks, max_queue_len, t_curr

def event_model(generator, processor, total_tasks=0, repeat=0):
    processed_tasks = 0
    t_gen = generator.generate()
    t_proc = t_gen + processor.generate()
    cur_queue_len = max_queue_len = 0
    reentered_tasks = 0
    while processed_tasks < total_tasks:
        if t_gen <= t_proc:
        cur_queue_len += 1
        if cur_queue_len > max_queue_len:
            max_queue_len = cur_queue_len
        t_gen += generator.generate()

    if t_gen >= t_proc:
        if cur_queue_len > 0:
            processed_tasks += 1
            if random.randint(1, 100) <= repeat:
                cur_queue_len += 1
                reentered_tasks += 1
            cur_queue_len -= 1
            if cur_queue_len > 0:
```

```
                t_proc += processor.generate()
            else:
                t_proc = t_gen + processor.generate()

    return processed_tasks, reentered_tasks, max_queue_len, t_proc
```

Листинг 3 – distributions.py

```
from numpy.random import uniform, normal, poisson
import random

class UniformDistribution:
    def __init__(self, a: float, b: float):
        self.a = a
        self.b = b

    def generate(self):
        return uniform(self.a, self.b)

class PoissonDistribution:
    def __init__(self, lyambda):
        self.lyambda = lyambda

    def generate(self):
        return poisson(self.lyambda)

class NormalDistribution:
    def __init__(self, mu, sigma):
        self.mu = mu
        self.sigma = sigma

    def generate(self):
        return normal(self.mu, self.sigma)
```