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 по курсу "Моделирование"

Тема Обработка очереди

Студент Леонов В.В.

Группа ИУ7-76Б

Оценка (баллы) _____

Преподаватель Рудаков И.В.

Москва — 2022 г.

Формальная постановка задачи

Промоделировать систему с пошаговым принципом и событийным принципом, определив минимальный размер буферной памяти, при котором не будет потерь сообщений.

Исходные данные:

- параметры генератора заявок;
- параметры обработчика заявок;
- приращение пошаговой модели;
- количество заявок;
- процент возврата заявок.

Краткие теоритические сведения

Генератор

Время прихода заявок из генератора описывается равномерным распределением.

Равномерное распределение — распределение случайной величины, принимающей значения, принадлежащей некоторому промежутку конечной длины, характрезующееся тем, что плотность вероятности в этом промежутке всюду постоянна.

Функция распределения равномерной непрерывной случайной величины имеет следующий вид:

$$F(x) = \begin{cases} 0, & x \leq a \\ \frac{x-a}{b-a}, & a \leq x \leq b \\ 1, & x > b \end{cases}$$

Плотность распределения равномерной непрерывной случайной величины имеет следующий вид:

$$f(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & \text{иначе} \end{cases}$$

Обработчик

Время обработки заявок из обработчика описывается распределением Эрланга.

Распределение Эрланга — распределение, описывающее непрерывную случайную величину, принимающую неотрицательные значения и представляющую собой сумму k независимых случайных величин, распределенных по одному и тому же экспоненциальному закону с параметром θ.

Функция распределения Эрланга непрерывной случайной величины имеет следующий вид:

$$F(x) = 1 - e^{-x/\theta} \sum_{i=0}^{k-1} \frac{(x/theta)^i}{i!}$$

Плотность распределения Эрланга непрерывной случайной величины имеет следующий вид:

$$f(x) = \frac{x^{k-1} e^{-x\theta} \theta^k}{(k-1)!}$$

Принципы моделирования работы системы

Пошаговый принцип

Заключается в последовательном анализе состояний всех блоков системы в момент t + Δt по заданному состоянию в момент t. При этом новое состояние блоков определяется в соответствии с их алгоритмическим описанием с учетом действующих случайных факторов. В результате этого анализа принимается решение о том, какие системные события должны имитироваться на данный момент времени. Основным недостатком являются значительные затраты машинных ресурсов, а при недостаточном малых Δt появляется опасность пропуска события.

Событийный принцип

Состояние отдельных устройств изменяется в дискретные моменты времени, совпадающие в моментами поступления сообщения, окончания решения задачи, возникновения аварийных сигналов и т. д. При использовании событийного принципа состояния всех боков системы анализируется лишь в момент появления какого-либо события. Момент наступления следующего события определяется минимальным значением из списка будущих событий, представляющий собой совокупность моментов ближайшего изменения состояния каждого из блоков. Момент наступления следующего события определяется минимальным значением из списка событий.

Средства реализации

Язык программирования — Python.
GUI — QT.

Листинг кода

```
import random

class TimeModel():
    def __init__(self, generator, processor, step: float) -> None:
        self.generator = generator
        self.processor = processor
        self.processed_tasks = 0
        self.step = step

    def start(self, total_tasks: int, repeat_rate: int) -> int:
        t_prev = 0
        t_current = self.step
        t_generator = self.generator.generate()
        t_processor = 0
        q_current_len = 0
        q_max_len = 0
        q_free = True

        while self.processed_tasks < total_tasks:
            if t_current > t_generator:
                q_current_len += 1
                if q_current_len > q_max_len:
                    q_max_len = q_current_len
                t_prev = t_generator
                t_generator += self.generator.generate()
            if t_current > t_processor:
                if q_current_len > 0:
                    q_prev = q_free
                    if q_free:
                        q_free = False
                    else:
                        self.processed_tasks += 1
                        if random.randint(0, 100) <= repeat_rate:
                            q_current_len += 1
                    q_current_len -=1
                    if q_prev:
                        t_processor = t_prev + self.processor.generate()
                    else:
                        t_processor += self.processor.generate()
                else:
                    q_free = True
            t_current += self.step
        return q_max_len
```

```
import random

class EventModel():
    def __init__(self, generator, processor) -> None:
        self.generator = generator
        self.processor = processor
        self.processed_tasks = 0

    def start(self, total_tasks: int, repeat_rate: int) -> int:
        events = Events()
        events.append(Event(self.generator.generate(), 'g'))
        q_current_len = 0
        q_max_len = 0
        q_free = True
        q_process = False

        while self.processed_tasks < total_tasks:
            e = events.pop(0)

            if e.type == 'g':
                q_current_len += 1
                if q_current_len > q_max_len:
                    q_max_len = q_current_len
                new_e = Event(e.time + self.generator.generate(), 'g')
                events.append(new_e)
                if q_free:
                    q_process = True

            elif e.type == 'p':
                self.processed_tasks += 1
                if random.randint(0, 100) <= repeat_rate:
                    q_current_len += 1
                q_process = True

            if q_process:
                if q_current_len > 0:
                    q_current_len -= 1
                    new_e = Event(e.time + self.processor.generate(), 'p')
                    events.append(new_e)
                    q_free = False
                else:
                    q_free = True
                q_process = False

        return q_max_len
```

```
1       class Event:
2           def __init__(self, time: float, type: str) -> None:
3               self.time = time
4               self.type = type
5
6
7       class Events:
8           def __init__(self) -> None:
9               self.__pool: list[Event] = []
10
11          def append(self, e: Event):
12              i = 0
13              while i < len(self.__pool) and self.__pool[i].time < e.time:
14                  i += 1
15              if 0 < i < len(self.__pool):
16                  self.__pool.insert(i - 1, e)
17              else:
18                  self.__pool.insert(i, e)
19
20          def pop(self, index):
21              return self.__pool.pop(index)
```

Демонстрация работы программы

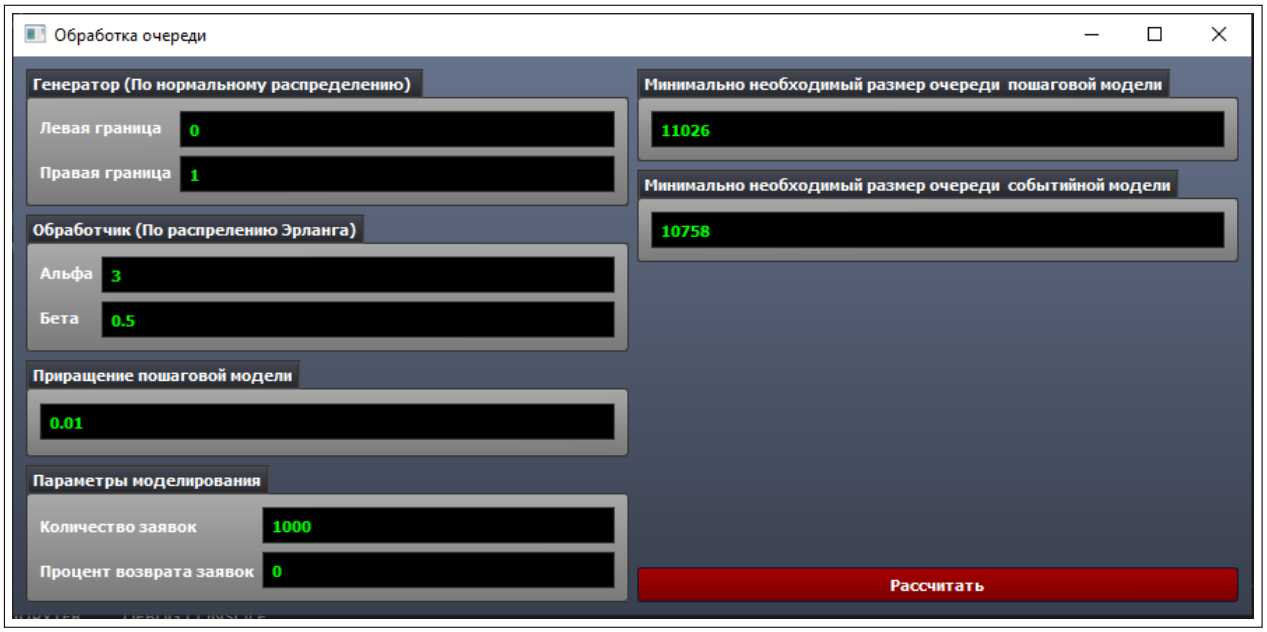

Рисунок 1 – Время обработки больше времени прихода (возврат = 0%)

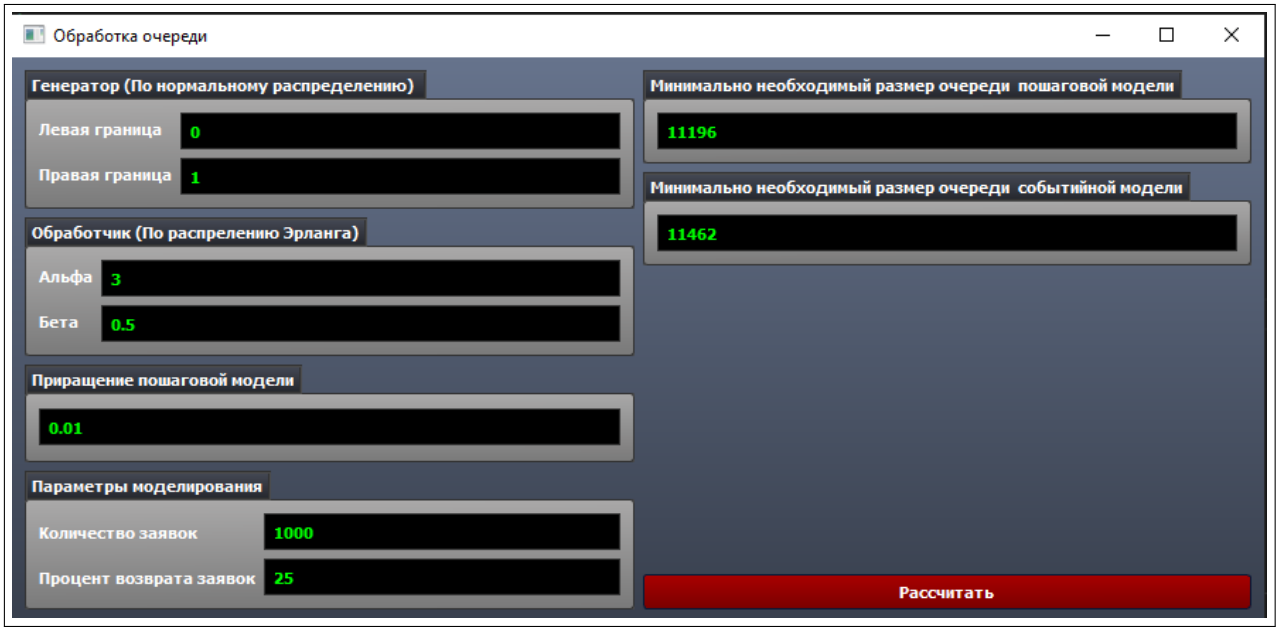

Рисунок 2 – Время обработки больше времени прихода (возврат = 25%)

Рисунок 3 – Время обработки больше времени прихода (возврат = 50%)

Рисунок 4 – Время обработки больше времени прихода (возврат = 75%)

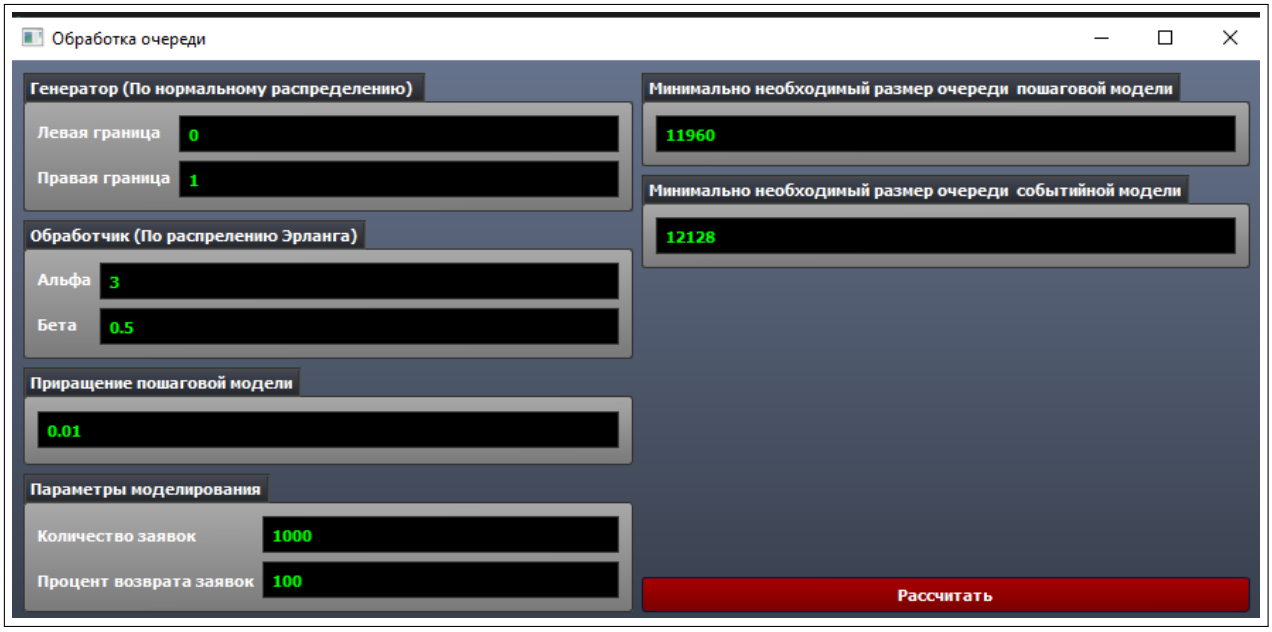

Рисунок 5 – Время обработки больше времени прихода (возврат = 100%)

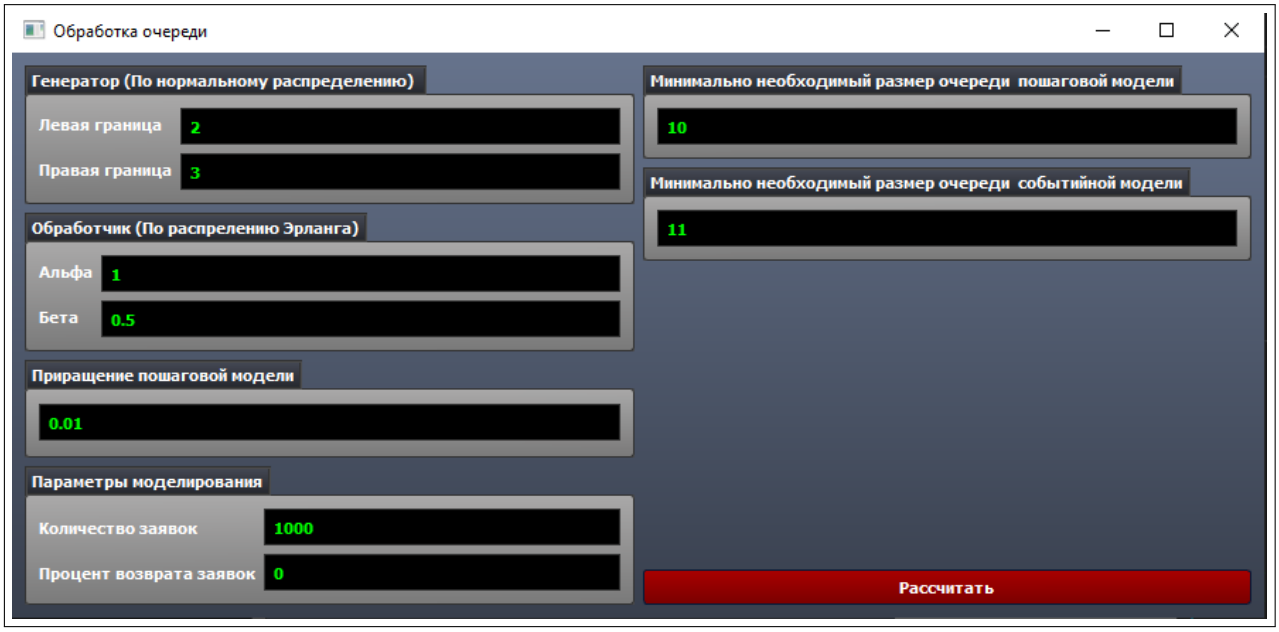

Рисунок 6 – Время обработки меньше времени прихода (возврат = 0%)

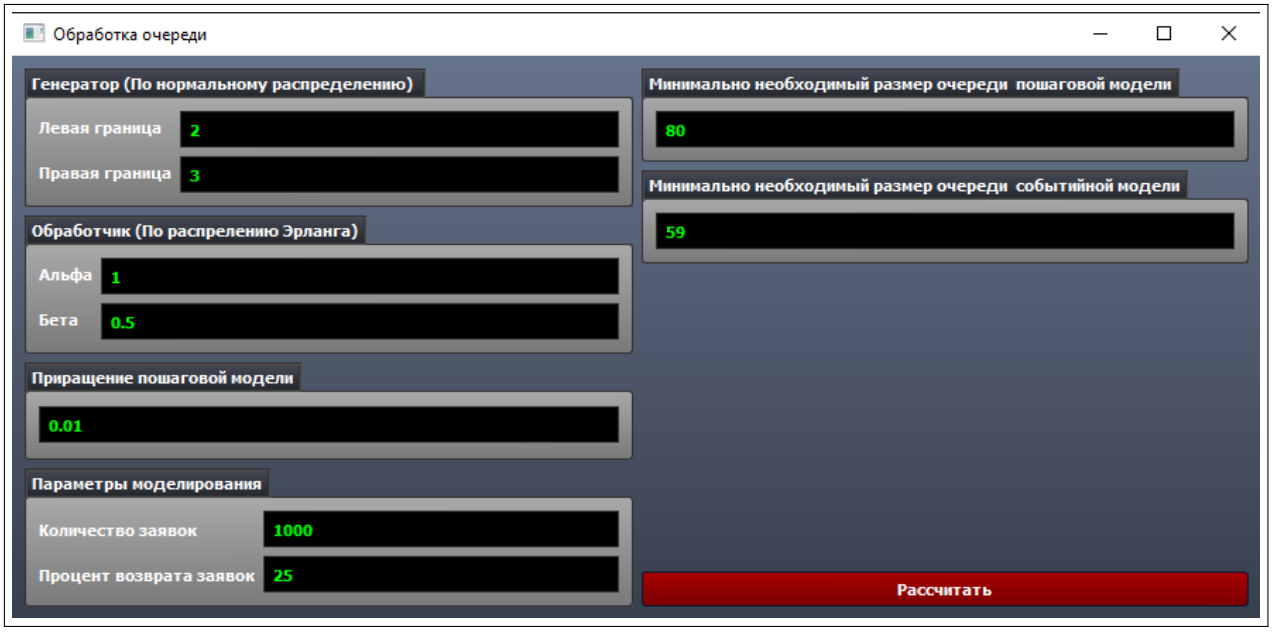

Рисунок 7 – Время обработки меньше времени прихода (возврат = 25%)

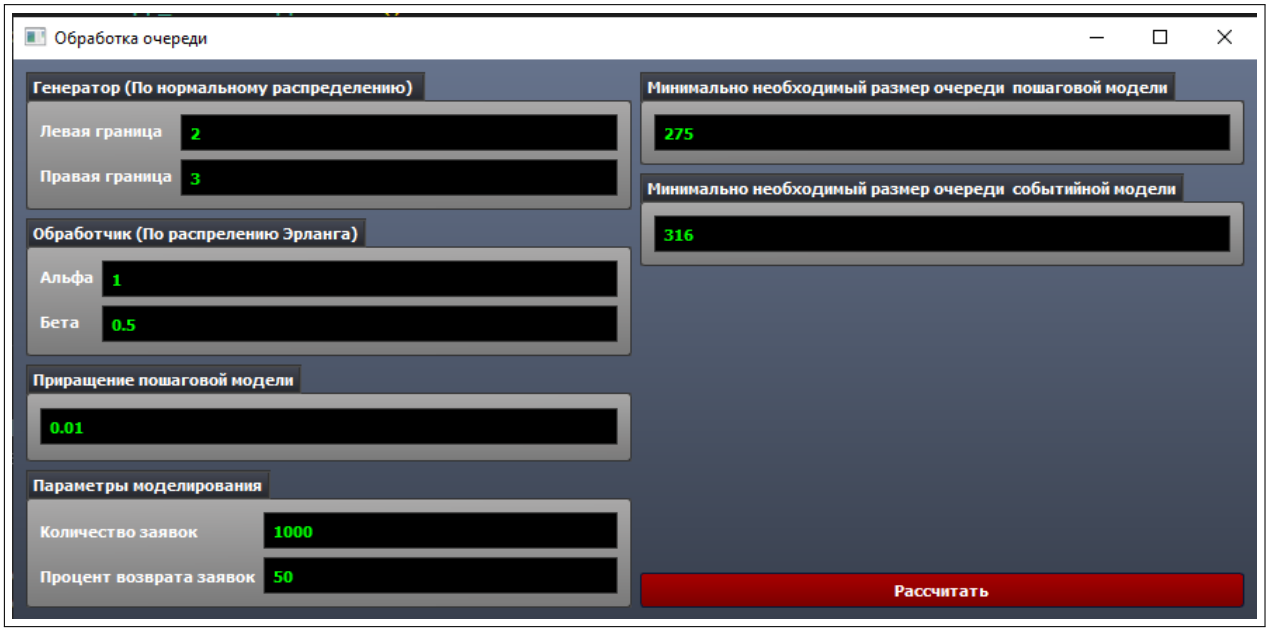

Рисунок 8 – Время обработки меньше времени прихода (возврат = 50%)

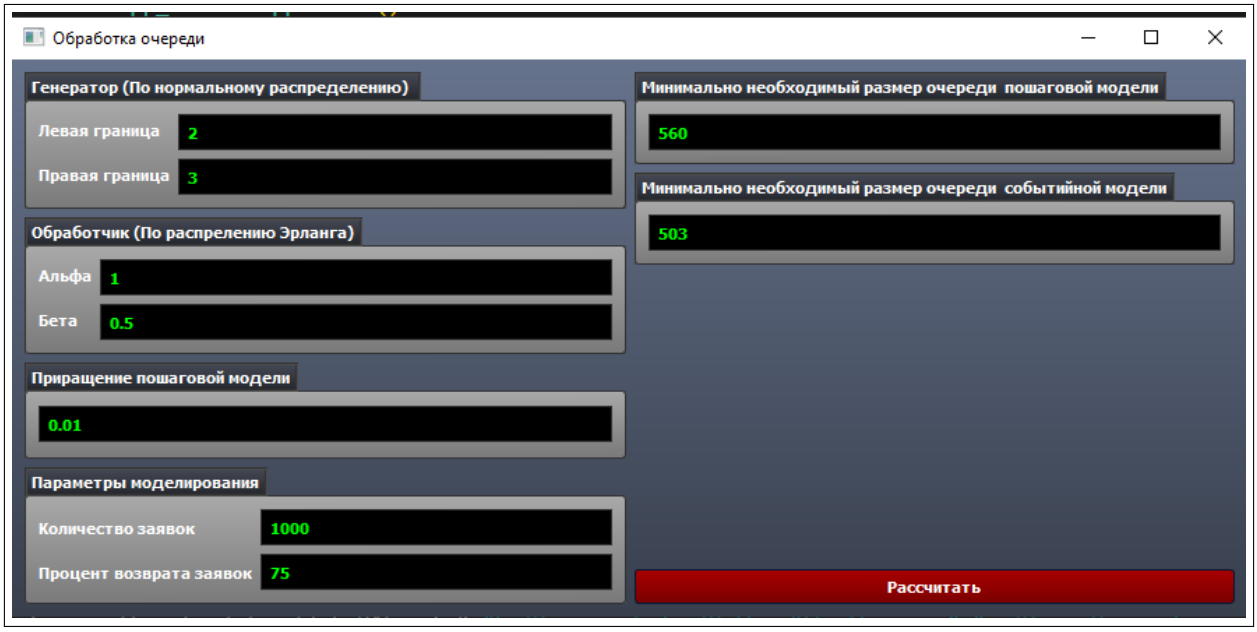

Рисунок 9 – Время обработки меньше времени прихода (возврат = 75%)

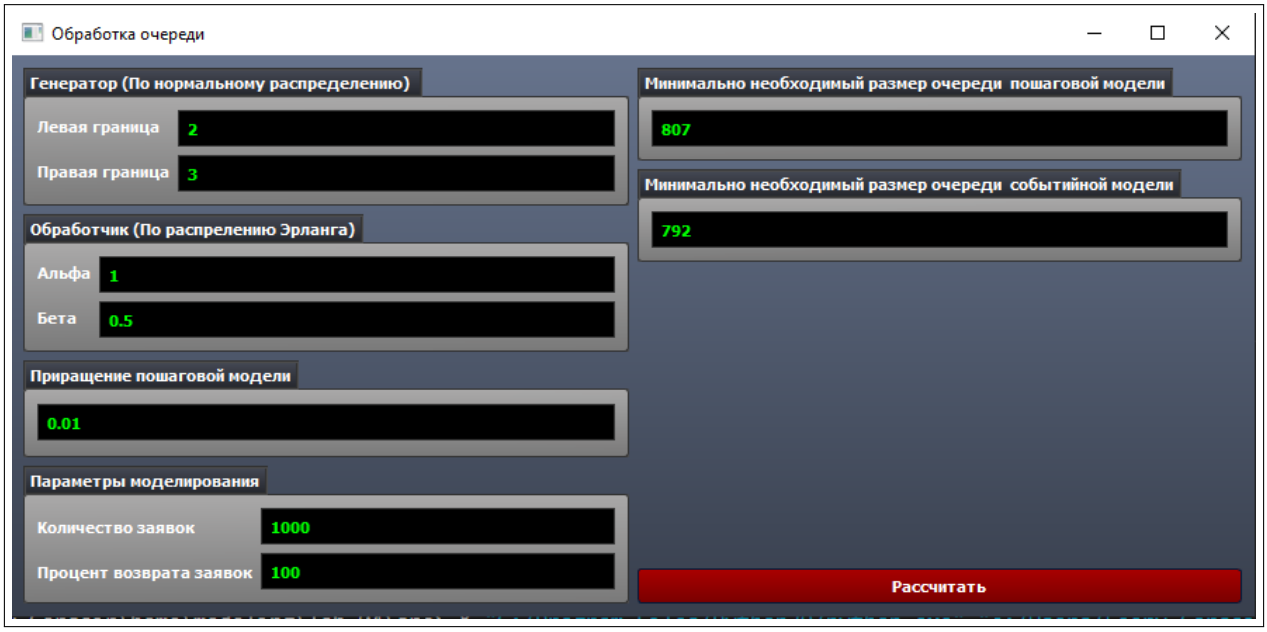

Рисунок 10 – Время обработки меньше времени прихода (возврат = 100%)