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 (ИУ)»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии (ИУ7)»

ОТЧЕТ

по лабораторной работе № 4

по курсу «Моделирование»

на тему: «Моделирование работы системы массового обслуживания»

Вариант № 2

Студент	ИУ7-73Б (Группа)	(Подпись, дата)	Р. Р. Хамзина (И. О. Фамилия)
Преподаватель		(Подпись, дата)	И. В. Рудаков (И. О. Фамилия)

2022 г.

СОДЕРЖАНИЕ

1 Задание

Реализовать программу с графическим интерфейсом для моделирования работы системы массового обслуживания принципом Δt и событийным принципом и определения максимальной длины очереди, при которой не будет потери сообщений. Моделируемая система состоит из генератора сообщений, очереди сообщений и обслуживающего аппарата. Для моделирования работы генератора сообщений использовать равномерный закон распределения, для моделирования работы обслуживающего аппарата — нормальный закон распределения. Предусмотреть возможность возврата в очередь части обработанных сообщений с заданной вероятностью.

1.1 Принцип Δt

Принцип Δt заключается в последовательном анализе состояний всех блоков в момент $t+\Delta t$ по заданному состоянию блоков в момент времени t. При этом новое состояние блоков объявляется в соответствии с их алгоритмическим описанием с учетом действующих случайных факторов, которые задаются распределениями вероятностей. В результате анализа принимается решение о том, какие общесистемные события должны имитироваться программой на данный момент времени.

1.2 Событийный принцип

При использовании событийного принципа состояния всех блоков имитационной модели анализируется лишь в момент появления какого-либо события. Момент поступления следующего события определяется минимальным значением из списка будущих событий, представляющего собой совокупность моментов ближайшего изменения состояний каждого из блоков системы.

1.3 Закон появления сообщений

Для моделирования работы генератора сообщений в лабораторной работе используется равномерный закон распределения. Случайная величина имеет равномерное распределение на отрезке $[a, b]$, если её функция плотности $p(x)$ имеет вид:

$$p(x) = \begin{cases} \frac{1}{b-a}, \text{если } x \in [a,b], \\ 0, \text{иначе.} \end{cases} \tag{1.1}$$

Функция распределения $F(x)$ равномерной случайной величины имеет вид:

$$F(x) = \begin{cases} 0, \text{если } x \leqslant a, \\ \frac{x-a}{b-a}, \text{если } a < x \leqslant b, \\ 1, \text{если } x > b. \end{cases} \tag{1.2}$$

Интервал времени между появлением i-ого и $(i-1)$-ого сообщения по равномерному закону распределения вычисляется следующим образом:

$$T_i = a + (b-a) \cdot R, \tag{1.3}$$

где R — псевдослучайное число от 0 до 1.

1.4 Закон обработки сообщений

Для моделирования работы генератора сообщений в лабораторной работе используется нормальный закон распределения. Случайная величина имеет нормальное распределение, если её функция плотности $p(x)$ имеет вид:

$$p(x) = \frac{1}{\sigma \cdot \sqrt{2 \cdot \pi}} \cdot e^{-\frac{(x-\mu)^2}{2 \cdot \sigma^2}}, (-\infty < \mu < +\infty, \sigma > 0). \tag{1.4}$$

Функция распределения $F(x)$ нормальной случайной величины имеет вид:

$$F(x) = \frac{1}{2} \cdot (1 + erf(\frac{x-\mu}{\sqrt{2 \cdot \sigma^2}})). \tag{1.5}$$

Интервал времени между появлением i-ого и $(i-1)$-ого сообщения по нормальному закону распределения вычисляется следующим образом:

$$T_i = \sigma_X \cdot \sqrt{\frac{12}{n}} \cdot (\sum_{i=1}^{n} R_i - \frac{n}{2}) + M_X, \tag{1.6}$$

где $n = 12$, R_i — псевдослучайное число от 0 до 1.

2 Реализация

2.1 Детали реализации

На листинге 2.1 показана реализация функции управляющей программы принципом Δt.

Листинг 2.1 – Реализация управляющей программы принципом Δt

```
def step_principle(self):
    max_length = 0
    now_length = 0
    processed_msgs = 0
    self.handler.free = True

    now_time = self.step
    generation_time = self.generator.get_time_interval()
    prev_generation_time = 0
    handling_time = 0

    while processed_msgs < self.msg_number:

        if now_time > generation_time:
            now_length += 1

            if max_length < now_length:
                max_length = now_length

            prev_generation_time = generation_time
            generation_time += self.generator.get_time_interval()

        if now_time > handling_time:
            if now_length > 0:
                was_free = self.handler.free

                if self.handler.free:
                    self.handler.free = False
                else:
                    processed_msgs += 1
                    now_length -= 1

                    now_return_probability = random()

```

```
                    if now_return_probability <= self.return_probability:
                        now_length += 1

                if was_free:
                    handling_time = prev_generation_time + \
                        self.handler.get_time_interval()
                else:
                    handling_time += self.handler.get_time_interval()

            else:
                self.handler.free = True

        now_time += self.step

    return max_length
```

На листинге 2.2 представлена реализация функции управляющей программы событийным принципом.

Листинг 2.2 – Реализация управляющей программы событийным принципом

```
def eventful_principle(self):
    max_length = 0
    now_length = 0
    processed_msgs = 0
    processed = False
    self.handler.free = True

    events = [[self.generator.get_time_interval(),
       State.generation]]

    while processed_msgs < self.msg_number:
        event = events.pop(0)

        if event[Constants.state] == State.generation:
            now_length += 1

            if max_length < now_length:
                max_length = now_length

            self.__add_event(events, [event[Constants.time] + \
                                 self.generator.get_time_interval(),
                                 State.generation])

            if self.handler.free:
                processed = True

        if event[Constants.state] == State.handling:
            processed_msgs += 1
            now_return_probability = random()

            if now_return_probability <= self.return_probability:
                now_length += 1

            processed = True

        if processed:
            if now_length > 0:
                now_length -= 1
```

```
                self.__add_event(events,
                                 [event[Constants.time] + \
                                 self.handler.get_time_interval(),
                                 State.handling])
                self.handler.free = False
            else:
                self.handler.free = True

            processed = False

    return max_length

def __add_event(self, events, event):
    i = 0

    while i < len(events) and \
            events[i][Constants.time] < event[Constants.time]:
            i += 1

    if 0 < i < len(events):
        events.insert(i - 1, event)
    else:
        events.insert(i, event)
```

На листинге 2.3 показана реализация функции вычисления интервала времени между появлением i-ого и $(i-1)$-ого сообщения по равномерному закону распределения.

Листинг 2.3 – Вычисление интервала времени между появлениями сообщений по равномерному закону распределения

```
def get_time_interval(self):
    return self.a + (self.b - self.a) * random()
```

На листинге 2.4 представлена реализация функции вычисления интервала времени между появлением i-ого и $(i-1)$-ого сообщения по нормальному закону распределения.

Листинг 2.4 – Вычисление интервала времени между появлениями сообщений по нормальному закону распределения

```
def get_time_interval(self):
    random_sum = sum([random() for _ in range(Constants.n)])
    return self.sigma * (random_sum - 6) + self.mu
```

2.2 Полученный результат

На рисунках 2.1-2.4 показаны страницы программы для определения максимальной длины очереди при моделировании системы массового обслуживания принципом Δt и событийным принципом с вероятностями возврата сообщения 0.0, 0.3, 0.7, 1.0 соответственно.

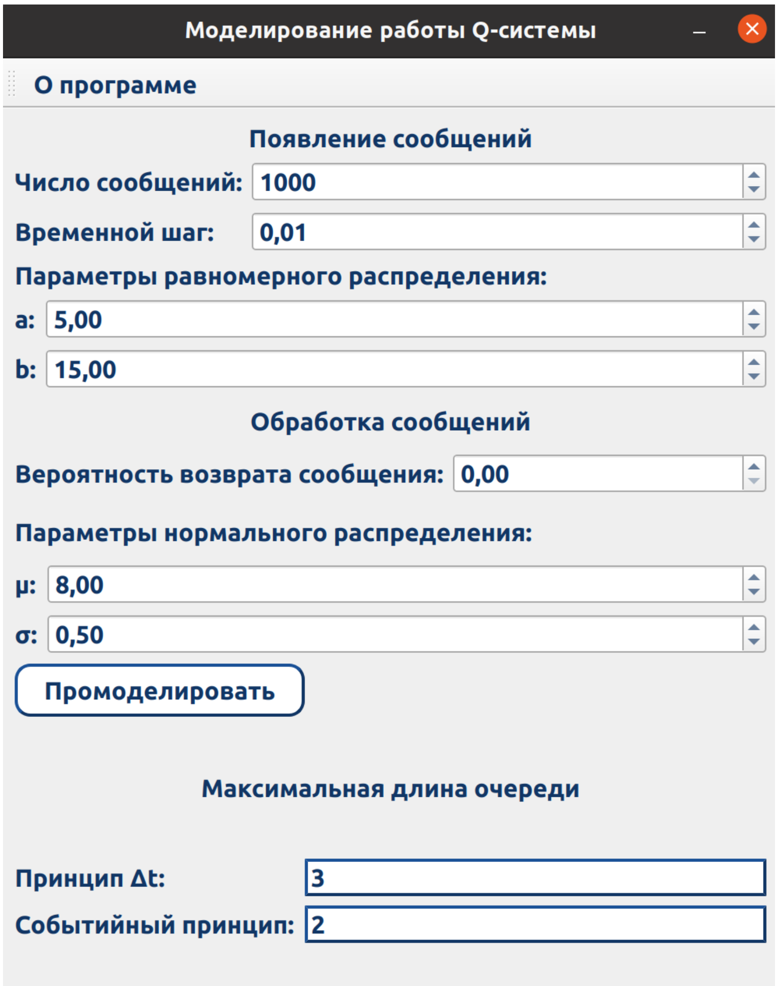

Рисунок 2.1 – Моделирование системы массового обслуживания с вероятностью возврата сообщения 0.0

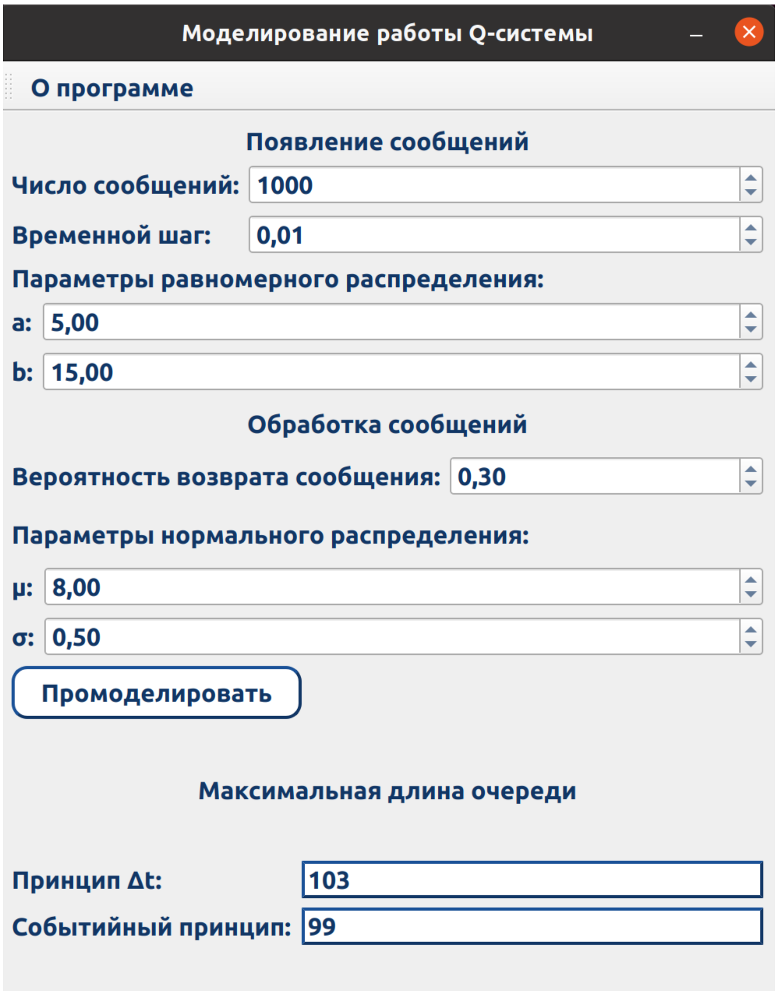

Рисунок 2.2 – Моделирование системы массового обслуживания с вероятностью возврата сообщения 0.3

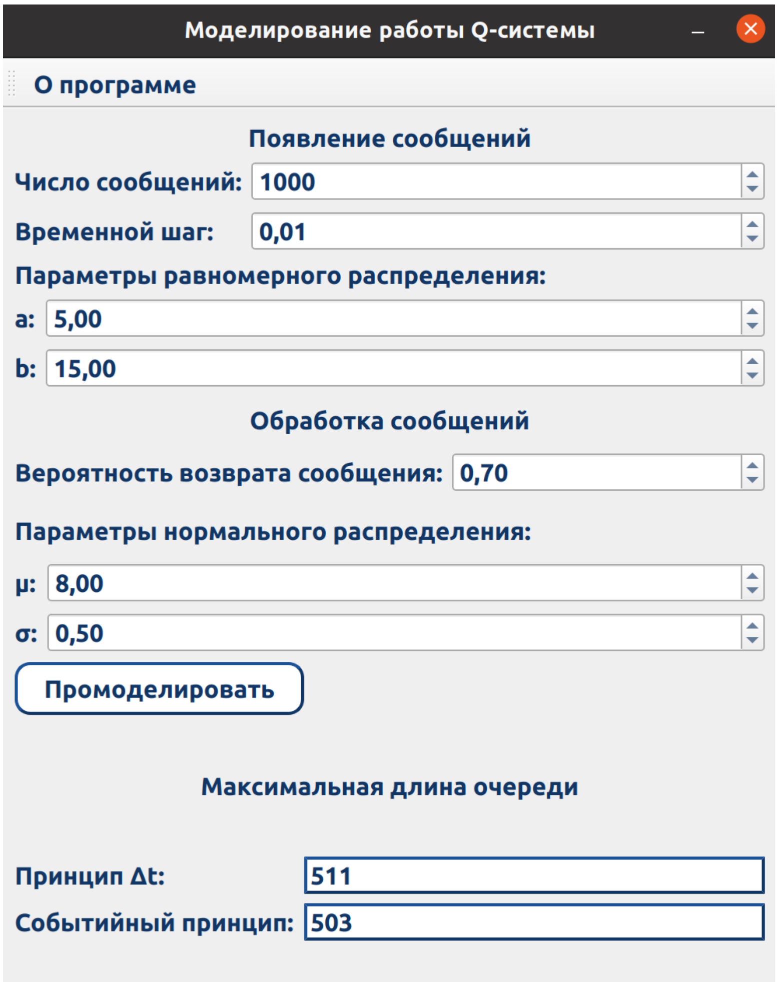

Рисунок 2.3 – Моделирование системы массового обслуживания с вероятностью возврата сообщения 0.7

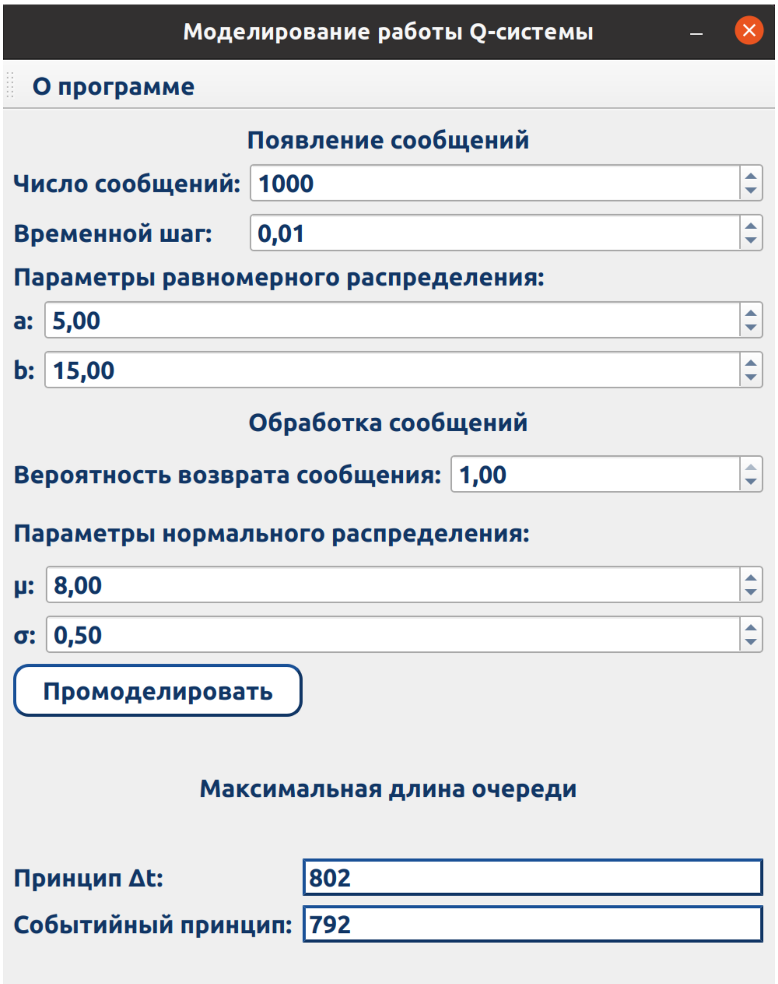

Рисунок 2.4 – Моделирование системы массового обслуживания с вероятностью возврата сообщения 1.0