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 Моделирование простейшей СМО

Студент Калашков П. А.

Группа ИУ7-76Б

Оценка (баллы)

Преподаватели Рудаков И. В.

Москва — 2023 г.

Целью данной работы является разработка программы с графическим интерфейсом для моделирования системы массового обслуживания (СМО) при помощи принципа Δt и событийного принципа и определения максимальной длины очереди, при которой не будет потери сообщений. Рассматриваемая СМО состоит из генератора сообщений, очереди ожидающих обработки сообщений и обслуживающего аппарата (ОА). Генерация сообщений происходит по равномерному закону распределения, время обработки сообщений — согласно закону распределения Эрланга. Необходимо предоставить возможности ручного задания необходимых параметров, а также возможности возврата обработанного сообщения в очередь обработки с заданной вероятностью.

Принципы работы управляющей программы модели

Принцип Δt

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

Данный принцип имеет недостаток: значительные затраты машинного времени на анализ и контрооль всей системы. Также при недостаточно малом Δt появляется опасность пропуска отдельных событий в системе, что исключает возмоожность молучения правильных результатов при моделировании.

Событийный принцип

Характерным свойством, отличающим событийный принцип, является изменение состояний свойств в дискретные моменты времени, совпадающие с моментами поступления сигналов окончания или аварийных сигналов. Состояния всех блоков программной (имитационной) модели анализируются лишь в момент появления события. Момент наступления нового события определяется минимальным значением из списка будущих событий, представляющих собой совокупность ближайшего изменения состояния каждого блока системы.

Используемые законы распределения

Закон появления сообщений

Согласно заданию лабораторной работы для генерации сообщений используется равномерный закон распределения. Случайная величина имеет равномерное распределение на отрезке $[a, b]$, если её функция плотности $p(x)$ имеет вид:

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

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

$$F(x) = \begin{cases} 0, \text{если } x < a, \\ \frac{x-a}{b-a}, \text{если } a < xb, \\ 1, \text{если } x > b. \end{cases} \tag{2}$$

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

$$T_i = a + (b - a) \cdot R, \tag{3}$$

где R — псевдослучайное число от 0 до 1.

Закон обработки сообщений

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

$$p(x) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!} \tag{4}$$

В распределении Эрланга целочисленный положительный параметр k — параметр формы (т. е. он влияет на форму распределения,а не просто сдвигает его, как параметр местоположения, или растягивает его или сжимает, как параметр масштаба), а параметр λ — параметр скорости (т. е. он обратен параметру масштаба, отвечающему за растягивание или сжатие графика распределения).

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

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

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

$$T_i = -\frac{1}{k\lambda} \sum_{j=1}^{k} ln(1 - R_i), \tag{5}$$

где R_j — псевдослучайное число от 0 до 1.

Результаты работы

Детали реализации

В листинге 1 представлена реализация управляющей программы принципа Δt, а в листинге 2 — реализация управляющей программы событийного принципа. Реализации функций вычисления интервала времени между появлениями сообщений по равномерному закону распределения и времени обработки сообщения поо закону распределения Эрланга представлены в листингах 3 и 4 соотвественно.

Листинг 1 – Реализация управляющей программы принципа Δt

```
def delta_t(self):
    max_length = 0
    queue_length = 0
    processed_amount = 0
    self.handler.free = True
    handling_time = 0
    current_time = self.step
    generated_time = self.generator.get_time_interval()
    previous_generated_time = 0
    while processed_amount < self.messages_amount:
        if current_time > generated_time:
            queue_length += 1
            if queue_length > max_length:
                max_length = queue_length
            previous_generated_time = generated_time
            generated_time += self.generator.get_time_interval()
        if current_time > handling_time:

            if queue_length > 0:
                handler_was_free = self.handler.free
                if self.handler.free:
                    self.handler.free = False
                else:
                    processed_amount += 1
                    queue_length -= 1
                    return_chance = random()
                    if return_chance <= self.return_chance:
                        queue_length += 1

                if handler_was_free:
                    handling_time = previous_generated_time +
                        self.handler.get_time_interval()
                else:
                    handling_time +=
                        self.handler.get_time_interval()
            else:
                self.handler.free = True
        current_time += self.step
    return max_length
```

Листинг 2 – Реализация управляющей программы событийного принципа

```
def event_driven(self):
        max_length = 0
        queue_length = 0
        processed_amounts = 0
        processed = False
        self.handler.free = True

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

        while processed_amounts < self.messages_amount:
            event = events.pop(0)

            if event[state] == generation:
                queue_length += 1
                if queue_length > max_length:
                    max_length = queue_length
                self.__insert_event(events, [event[time] +
                   self.generator.get_time_interval(), generation])
                if self.handler.free:
                    processed = True

            if event[state] == handling:
                processed_amounts += 1
                return_chance = random()
                if return_chance <= self.return_chance:
                    queue_length += 1
                processed = True
            if processed:
                if queue_length > 0:
                    queue_length -= 1
                    self.__insert_event(events, [event[time] +
                       self.handler.get_time_interval(), handling])
                    self.handler.free = False
                else:
                    self.handler.free = True
                processed = False
        return max_length
```

Листинг 3 – Вычисление интервала времени между появлениями сообщений по равномерному закону распределения

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

Листинг 4 – Вычисление времени обработки сообщения по закону распределения Эрланга

```
def get_time_interval(self):
        random_sum = sum([log(1 - random()) for _ in range(self.k)])
        chance = -1 / (self.k * self.lambd) * random_sum
        print(chance)
        return chance
```

Примеры работы

На рисунках 1 и 2 представлены примеры работы разработанной программы для описанной СМО без возврата обработанных сообщений и с возвратом.

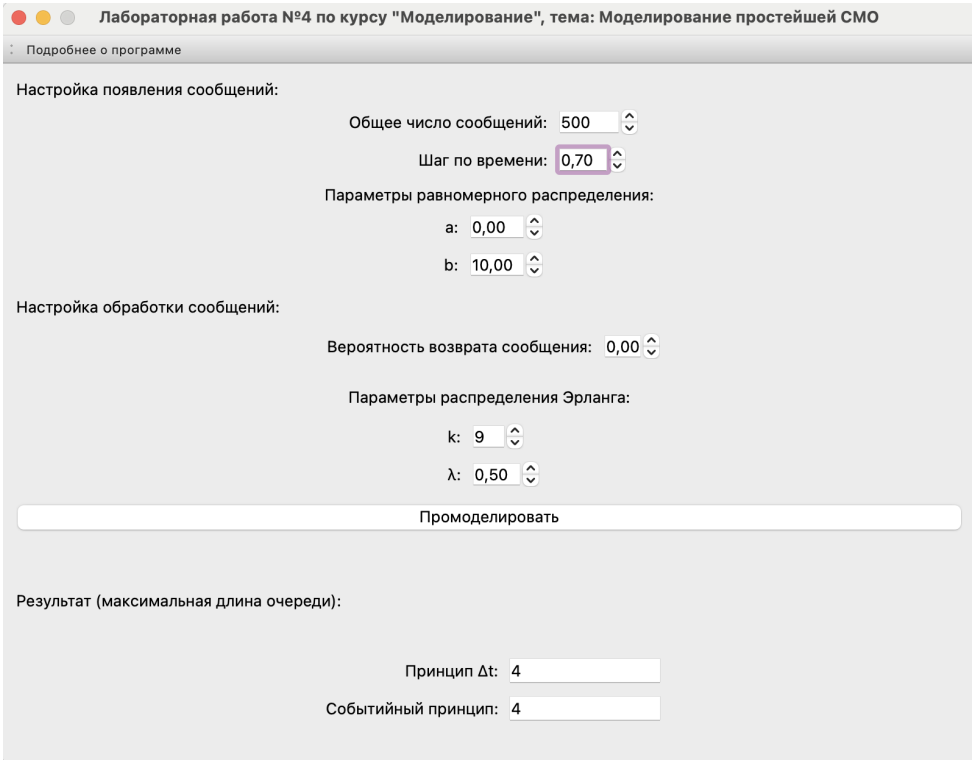

Рисунок 1 – Пример работы программы, введённые вручную значения равны единице

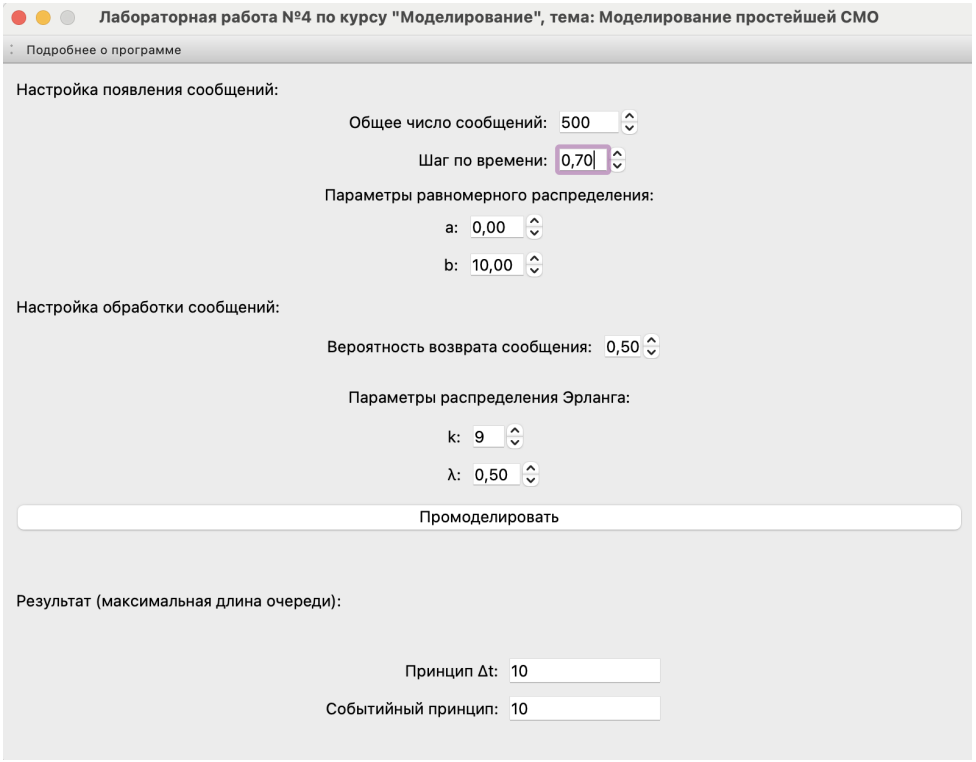

Рисунок 2 – Пример работы программы, введённые вручную значения отличаются друг от друга на константу

Вывод

В ходе выполнения лабораторной работы была реализована программа с графическим интерфейсом для моделирования системы массового обслуживания (СМО) при помощи принципа Δt и событийного принципа и определения максимальной длины очереди, при которой не будет потери сообщений.