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 Фам Минь Хиеу

Группа ИУ7И-72Б

Оценка (баллы) ______

Преподаватели Рудаков И. В.

Москва — 2024 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 нормального распределения. Необходимо предоставить возможности ручного задания необходимых параметров, а также возможности возврата обработанного сообщения в очередь обработки с заданной вероятностью.

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 нормальное распределение. Нормальное распределение - распределение вероятностей, которое в одномерном случае задаётся функцией плотности вероятности, совпадающей с функцией Гаусса:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\dfrac{(x-\mu)^2}{2\sigma^2}} \tag{4}$$

где параметр μ — математическое ожидание (среднее значение) распределения, а параметр σ - среднеквадратическое отклонение (σ^2 - дисперсия) распределения.

Функция распределения имеет вид:

$$F_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \int\limits_{-\infty}^{x} e^{-\dfrac{(x-\mu)^2}{2\sigma^2}}\, dx \tag{5}$$

Обозначают нормальное распределение $X \sim N(\mu, \sigma^2)$.

Стандартным нормальным распределением называется нормальное распределение с математическим ожиданием $\mu = 0$ и стандартным отклонением $\sigma = 1$.

Результаты работы

Пошаговый подход

Заключается в последовательном анализе состояний всех блоков системы в момент $t + \Delta t$ 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. Основной недостаток: значительные затраты машинных ресурсов, а при недостаточном малых Δt появляется опасность пропуска события.

Событийный принцип

Характерное свойство модели системы обработки информации: состояние отдельных устройств изменяется в дискретные моменты времени, совпадающие с 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.

Листинг 1 – Реализация управляющей программы принципа Δt

```
def step_model(generator, processor, total_tasks=0, repeat=0,
    step=0.001):
    processed_tasks = 0
    t_curr = step
    t_gen = generator.generate()
    t_gen_prev = t_proc = 0
    cur_queue_len = max_queue_len = 0
    free = True

    while processed_tasks < total_tasks:
        if t_curr > t_gen:
            cur_queue_len += 1

            if cur_queue_len > max_queue_len:
                max_queue_len = cur_queue_len

            t_gen_prev = t_gen
            t_gen += generator.generate()

        if t_curr > t_proc:
            if cur_queue_len > 0:
                was_free = free
                if free:
                    free = False
                else:
                    processed_tasks += 1
                    if random.randint(1, 100) <= repeat:
                        cur_queue_len += 1
                cur_queue_len -= 1
                if was_free:
                    t_proc = t_gen_prev + processor.generate()
                else:
                    t_proc += processor.generate()
            else:
                free = True
        t_curr += step

    return max_queue_len
```

Листинг 2 – Реализация управляющей программы событийного принципа

```
def event_model(generator, processor, total_tasks=0, repeat=0):
    processed_tasks = 0
    cur_queue_len = max_queue_len = 0
    events = [[generator.generate(), 'g']]
    free, process_flag = True, False

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

        if event[1] == 'g':
            cur_queue_len += 1
            if cur_queue_len > max_queue_len:
                max_queue_len = cur_queue_len
            add_event(events, [event[0] + generator.generate(),
                'g'])
            if free:
                process_flag = True

        elif event[1] == 'p':
            processed_tasks += 1
            if randint(1, 100) <= repeat:
                cur_queue_len += 1
            process_flag = True

        if process_flag:
            if cur_queue_len > 0:
                cur_queue_len -= 1
                add_event(events, [event[0] + processor.generate(),
                    'p'])
                free = False
            else:
                free = True
            process_flag = False

    return max_queue_len
```

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

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

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

Листинг 4 – Вычисление времени обработки сообщения по нормальному закону распределения

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

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

Примеры работы

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

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.

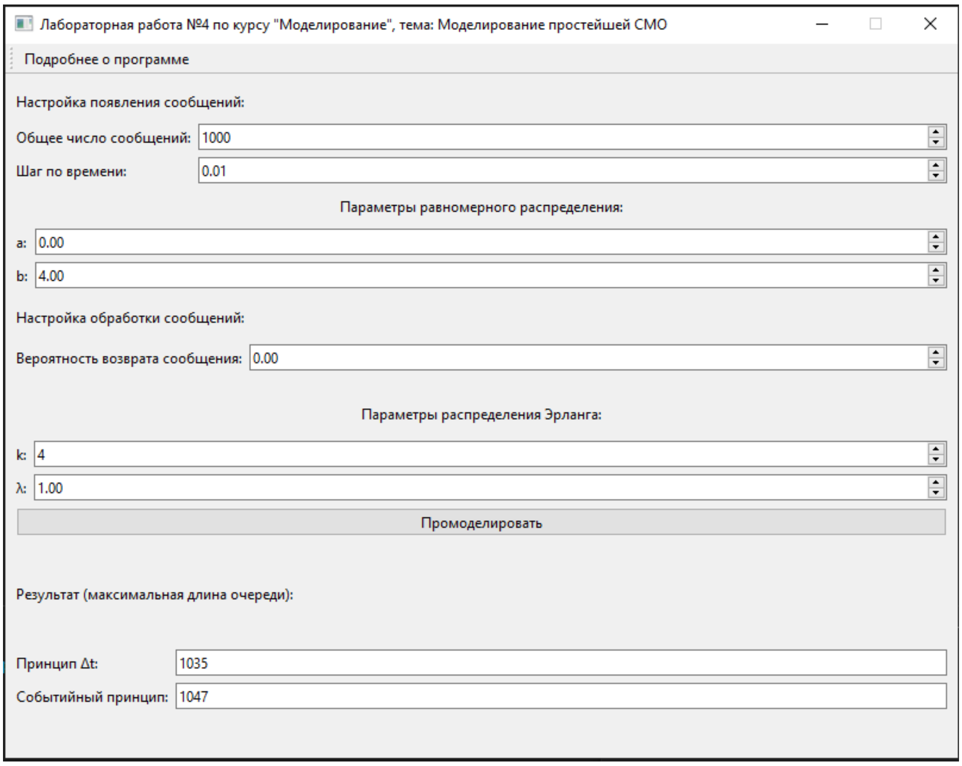

Рисунок 1 – Генератор создает сообщения интенсивнее, чем их обрабатывает автомат (при таких параметрах возникает переполнение)

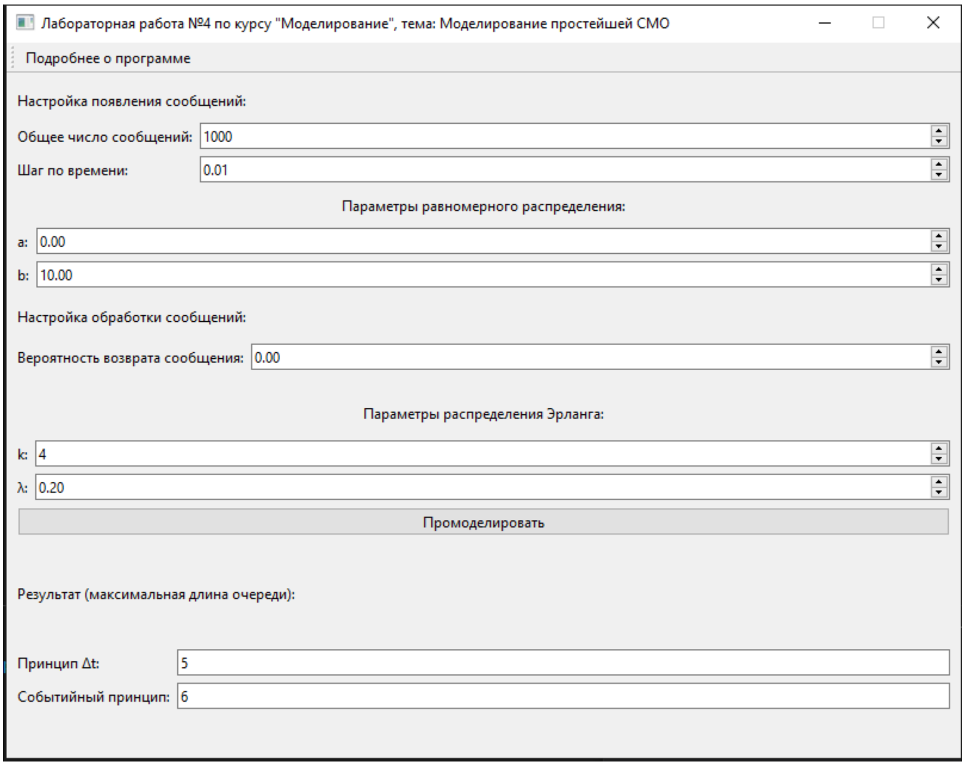

Рисунок 2 – Автомат обрабатывает сообщения интенсивнее, чем их создаёт генератор