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»

ОТЧЕТ

к лабораторной работе №3

По курсу: «Моделирование»

Студент ***ИУ7И-76Б*** _____ **Нгуен Ф. С.**
(Группа) (Подпись, дата) (И.О. Фамилия)

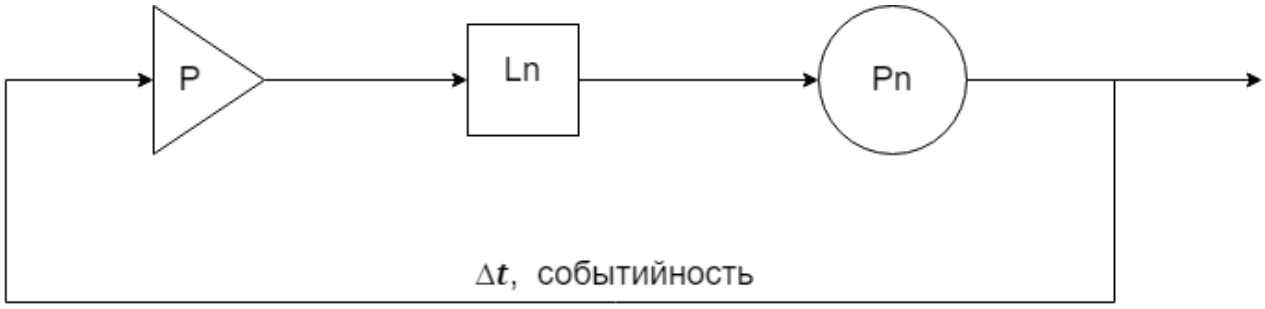
Преподаватель _____ **Рудаков И.В.**
(Подпись, дата) (И.О. Фамилия)

Москва, 2021 г.

Оглавление

I. Теоретическая часть

Смоделировать систему, состоящую из генератора, очереди и ОА.



Равномерное распределение:

$X \sim R(a, b)$, где $a, b \in$ R.

Функция распределения равномерной непрерывной случайной величины:

$$F(x) = \begin{cases} 0 & \text{при } x \le a \\ \dfrac{x-a}{b-a} & \text{при } a \le x \le b \\ 1 & \text{при } x > b \end{cases} \quad (1)$$

Плотность распределения равномерной непрерывной случайной величины:

$$f(x) = \begin{cases} \dfrac{1}{b-a} & \text{при } a \le x \le b \\ 0 & \text{иначе} \end{cases} \quad (2)$$

Нормальное распределение:

$X \sim N(\mu, \sigma^2)$, μ — математическое ожидание, σ^2 - дисперсия

Плотность распределения равномерной непрерывной случайной величины:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (3)$$

Функция распределения:

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx$$	*(4)*

Принцип Δt .

Принцип Δt заключается в последовательном анализе состояний всех блоков в момент **t +** Δt по заданному состоянию блоков в момент **t.** При этом новое состояние блоков определяется в соответствии с их алгоритмическим описанием с учетом действующих случайных факторов, задаваемых распределениями вероятности. В результате такого анализа принимается решение о том, какие общесистемные события должны имитироваться программной моделью на данный момент времени.

Основной недостаток этого принципа: значительные затраты машинного времени на реализацию моделирования системы. А при недостаточно малом Δt появляется опасность пропуска отдельных событий в системе, что исключает возможность получения адекватных результатов при моделировании.

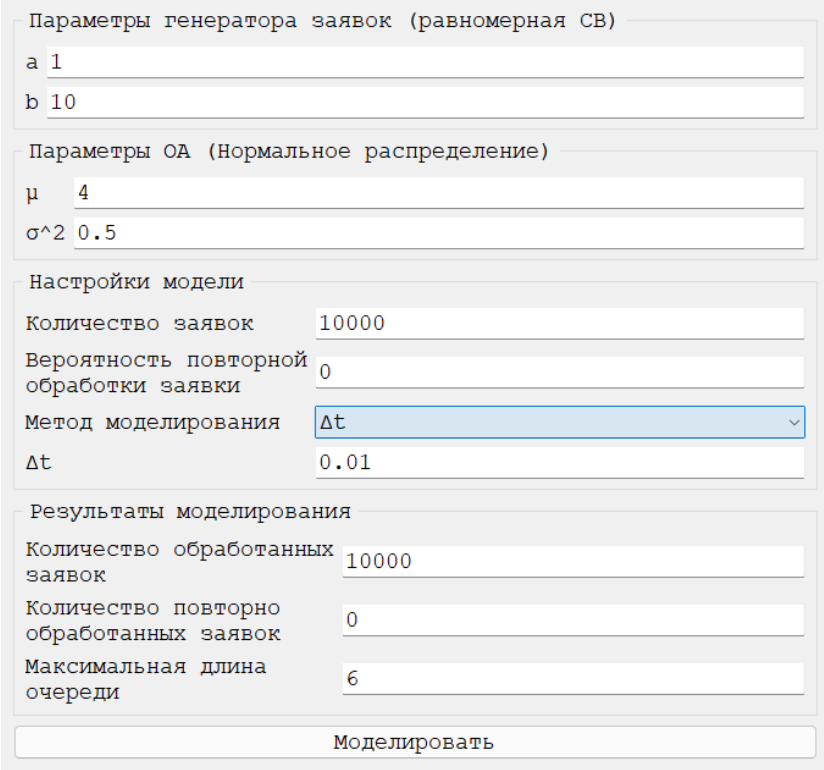
Достоинство: равномерная протяжка времени.

Событийный принцип.

Характерное свойство систем обработки информации то, что состояние отдельных устройств изменяются в дискретные моменты времени, совпадающие с моментами времени поступления сообщений в систему, временем поступления окончания задачи, времени поступления аварийных сигналов и т.д. Поэтому моделирование и продвижение времени в системе удобно проводить, используя **событийный принцип**, при

котором состояние 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 состояния каждого из блоков системы

II. Экспериментальная часть



Входные данные:

Количество заявок = 10000			
$R(a, b)$		$N(\mu, \sigma^2)$	
a	b	μ	σ^2
1	10	4	0.5

Выходные данные:

	$\Delta t = 0.01$		**Событийный**	
Вероятность повтор. обраб. заявки	Количество повторно обраб. заявок	Максимальная длина очереди	Количество повторно обраб. заявок	Максимальная длина очереди
0%	0	6	0	5
10%	991	11	1020	9
20%	2004	14	1994	14
50%	4977	2181	4951	2218
100%	10000	7278	10000	7261

III. Код программы:

```
class Modeller:
    def __init__(self, uniform_a, uniform_b, normal_mu, normal_sigma, reenter_prop):
        self._generator = RequestGenerator(UniformGenerator(uniform_a, uniform_b))
        self._processor = RequestProcessor(ErlangGenerator(normal_mu, normal_sigma),
reenter_prop)
        self._generator.add_receiver(self._processor)

    def event_based_modelling(self, request_count):
        generator = self._generator
        processor = self._processor

        gen_period = generator.next_time_period()
        proc_period = gen_period + processor.next_time_period()
        while processor.processed_requests < request_count:
            if gen_period <= proc_period:
                generator.emit_request()
                gen_period += generator.next_time_period()
            else:
                processor.process()
                if processor.current_queue_size > 0:
                    proc_period += processor.next_time_period()
                else:
                    proc_period = gen_period + processor.next_time_period()

        return (processor.processed_requests, processor.reentered_requests,
                processor.max_queue_size, proc_period)

    def time_based_modelling(self, request_count, dt):
        generator = self._generator
        processor = self._processor

        gen_period = generator.next_time_period()

        proc_period = gen_period + processor.next_time_period()
        current_time = 0
        while processor.processed_requests < request_count:
            if gen_period <= current_time:
                generator.emit_request()
                gen_period += generator.next_time_period()
            if current_time >= proc_period:
                processor.process()
                if processor.current_queue_size > 0:
                    proc_period += processor.next_time_period()
                else:
                    proc_period = gen_period + processor.next_time_period()
            current_time += dt

        return processor.processed_requests, processor.reentered_requests,
processor.max_queue_size, current_time
```