|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 №4*

*По курсу: «Моделирование»*

*Тема:* ***«Моделирование прибора обслуживания»***

Студент ИУ7-74Б

Жабин Д.В.

Преподаватель

Рудаков И.В.

*Москва, 2022 г.*

# Задание

Необходимо промоделировать систему, состоящую из генератора, памяти, и обслуживающего аппарата. Генератор подает сообщения, распределенные по равномерному закону, они поступают в память и выбираются на обработку по нормальному закону. Количество заявок конечно и задано. Предусмотреть случай, когда обработанная заявка с заданной вероятностью возвращается обратно в очередь. Необходимо определить минимальную длину очереди, при которой не будет возникать потерь сообщений. Реализовать двумя способами: используя пошаговый и событийный подходы.

# Теоретическая часть

## Равномерное распределение

Равномерное распределение — распределение случайной величины, принимающей значения, принадлежащие некоторому промежутку конечной длины, характеризующееся тем, что плотность вероятности на этом промежутке всюду постоянна.

Равномерное распределение обозначают 𝑋 ~ 𝑅(𝑎, 𝑏), где 𝑎, 𝑏 ∈ R.

Функция распределения равномерной непрерывной случайной величины:

Плотность распределения равномерной непрерывной случайной величины:

## Нормальное распределение

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

где параметр μ — математическое ожидание (среднее значение), медиана и мода распределения, а параметр σ — среднеквадратическое отклонение (σ2 — дисперсия) распределения.

Функция распределения:

Обозначают нормальное распределение 𝑋 ~ 𝑁(𝜇, 𝜎2).

Стандартным нормальным распределением называется нормальное распределение с математическим ожиданием μ = 0 и стандартным отклонением σ = 1.

Математическое ожидание μ характеризует положение «центра тяжести» вероятностной массы нормального распределения. Получается, что график плотности распределения случайной величины, имеющей нормальное распределение, симметричен относительно 𝑥 = 𝜇. Дисперсия σ характеризует разброс значений случайной величины относительно «центра тяжести».

## Принцип

Принцип заключается в последовательном анализе состояний всех блоков системы в момент по заданному состоянию блоков в момент . Новое состояние блоков определяется в соответствии с их алгоритмическим описанием с учетом действия случайных факторов, задаваемых распределениями вероятности. В результате этого анализа принимается решение о том, какие системные события должны имитироваться программной моделью на данный момент времени.

Основной недостаток этого принципа: значительные затраты машинного времени и опасность пропуска отдельных событий в системе при недостаточно малых .

## Событийный принцип

При использовании событийного принципа состояние всех блоков системы анализируется лишь в момент проявления какого-либо события. Моменты наступления следующего события определяются минимальным значением из списка будущих событий, представляющих собой совокупность моментов ближайшего изменения состояния каждого из блоков.

Проблема пропуска отдельных событий системы при использовании событийного принципа отсутствует, однако недостаток метода заключается в том, что при большом количестве событий системы список необходимо просматривать постоянно.

**Текст программы**

|  |
| --- |
| **from numpy.random import normal**  **from random import random**  **class EvenDistribution:**  **def \_\_init\_\_(self, a, b):**  **self.a = a**  **self.b = b**  **def generate(self):**  **return self.a + (self.b - self.a) \* random()**  **class NormalDistribution:**  **def \_\_init\_\_(self, mu, sigma):**  **self.mu = mu**  **self.sigma = sigma**  **def generate(self):**  **return normal(self.mu, self.sigma)**  **def step\_model(generator, processor, total\_tasks=0, repeat=0, step=0.001):**  **processed\_tasks = 0**  **t\_cur = step**  **t\_gen = generator.generate()**  **t\_proc = 0**  **cur\_qlen = max\_qlen = 0**  **generated\_tasks = 0**  **repeated\_tasks = 0**  **while processed\_tasks < total\_tasks + repeated\_tasks:**  **if t\_cur > t\_gen and generated\_tasks <= total\_tasks:**  **cur\_qlen += 1**  **generated\_tasks += 1**  **if cur\_qlen > max\_qlen:**  **max\_qlen = cur\_qlen**  **t\_gen += generator.generate()**  **if t\_cur > t\_proc and cur\_qlen > 0:**  **processed\_tasks += 1**  **if random() <= repeat:**  **repeated\_tasks += 1**  **cur\_qlen += 1**  **cur\_qlen -= 1**  **t\_proc += processor.generate()**  **t\_cur += step**  **return max\_qlen, processed\_tasks, repeated\_tasks**  **def add\_event(events, event):**  **i = 0**  **while i < len(events) and events[i][0] < event[0]:**  **i += 1**  **events.insert(i, event)**  **def event\_model(generator, processor, total\_tasks=0, repeat=0):**  **processed\_tasks = 0**  **cur\_qlen = max\_qlen = 0**  **events = [[generator.generate(), 'g']]**  **free, process\_flag = True, False**  **generated\_tasks = 0**  **repeated\_tasks = 0**  **while processed\_tasks < total\_tasks + repeated\_tasks:**  **event = events.pop(0)**  **if event[1] == 'g' and generated\_tasks <= total\_tasks:**  **cur\_qlen += 1**  **generated\_tasks += 1**  **if cur\_qlen > max\_qlen:**  **max\_qlen = cur\_qlen**  **add\_event(events, [event[0] + generator.generate(), 'g'])**  **if free:**  **process\_flag = True**  **elif event[1] == 'p':**  **processed\_tasks += 1**  **if random() <= repeat:**  **repeated\_tasks += 1**  **cur\_qlen += 1**  **process\_flag = True**  **if process\_flag:**  **if cur\_qlen > 0:**  **cur\_qlen -= 1**  **add\_event(events, [event[0] + processor.generate(), 'p'])**  **free = False**  **else:**  **free = True**  **process\_flag = False**  **return max\_qlen, processed\_tasks, repeated\_tasks**  **def main():**  **a, b = map(float, input('Введите границы диапазона для равномерного распределения: ').split())**  **generator = EvenDistribution(a, b)**  **mu, sigma = map(float, input('Введите параметры нормального распределения: ').split())**  **processor = NormalDistribution(mu, sigma)**  **tasks = int(input('Введите кол-во заявок в системе: '))**  **repeat\_prob = float(input('Укажите вероятность необходимости повторной обработки заявок: '))**  **step = 0.01**  **event\_res = event\_model(generator, processor, tasks, repeat\_prob)**  **deltaT\_res = step\_model(generator, processor, tasks, repeat\_prob, step)**  **print()**  **print('-----------Событийный принцип-----------')**  **print('Максимальная длина очереди:', event\_res[0], '\nОбработано заявок:', event\_res[1], '(', event\_res[2], 'повторно)')**  **print('-------------Принцип deltaT-------------')**  **print('Максимальная длина очереди:', deltaT\_res[0], '\nОбработано заявок:', deltaT\_res[1], '(', deltaT\_res[2], 'повторно)')**  **if \_\_name\_\_ == '\_\_main\_\_':**  **main()** |

# Примеры работы

## На рисунках 1-9 приведены результаты работы смоделированной системы при различных входных параметрах.

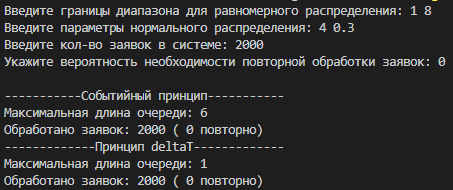


Рисунок 1 – 2000 заявок, без повторов

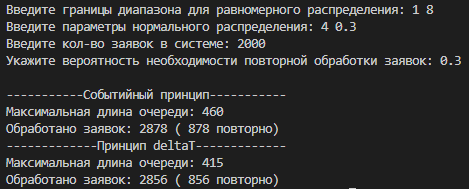


Рисунок 2 – 2000 заявок, 30% повторов

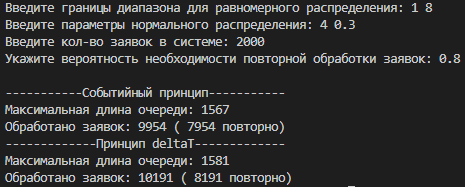


Рисунок 3 – 2000 заявок, 80% повторов

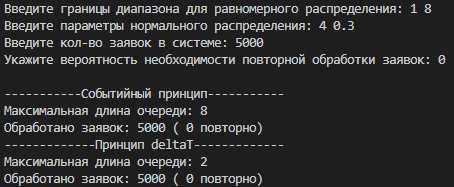


Рисунок 4 – 5000 заявок, без повторов

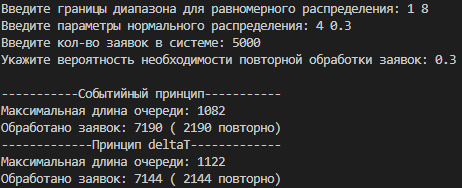


Рисунок 5 – 5000 заявок, 30% повторов

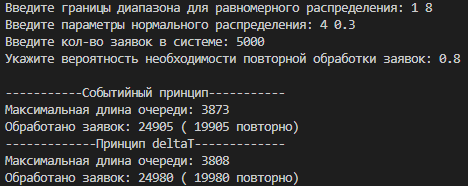


Рисунок 6 – 5000 заявок, 80% повторов

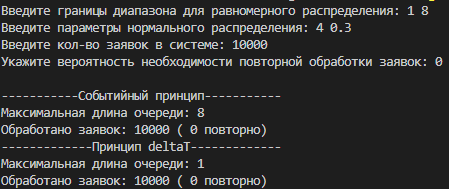


Рисунок 7 – 10000 заявок, без повторов

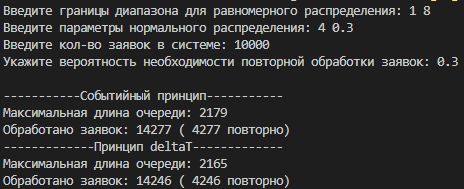


Рисунок 8 – 10000 заявок, 30% повторов

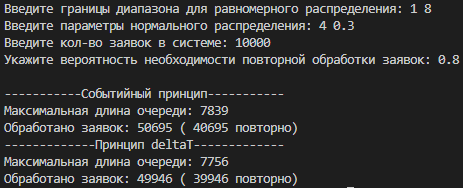


Рисунок 9 – 10000 заявок, 100% повторов